7-1-2007

Architecture design of a scalable adaptive deblocking filter for H.264/AVC

Eric Ernst

Architecture Design of a Scalable Adaptive Deblocking Filter for H.264/AVC

by

Eric Gerard Ernst

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Supervised by

Dr. Marcin Łukowiak
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
July 2007

Approved By:

Dr. Marcin Łukowiak
Assistant Professor, RIT Department of Computer Engineering
Primary Adviser

Dr. Andreas Savakis
Professor, RIT Department of Computer Engineering

Dr. Dhireesha Kudithipudi
Assistant Professor, RIT Department of Computer Engineering

Thesis Release Permission Form

Rochester Institute of Technology

Kate Gleason College of Engineering

Title: Architecture Design of a Scalable Adaptive Deblokcing Filter for H.264/AVC

I, Eric Gerard Ernst, hereby grant permission to the Wallace Memorial Library reporduce my thesis in whole or part.

Eric Gerard Ernst

Date

Dedication

This thesis is dedicated to my parents for their pride and support; and to my wife, whose patience and dedication helped push me through.

Acknowledgments

Thanks to Dr. Marcin Lukowiak for his constant advice and patience.

Abstract

Due to significant bit-rate savings and improved perceptual quality, H.264/AVC, the latest video compression standard from the Joint Video Team, is receiving widespread adoption. Greater coding efficiency relative to previous standards is a result of additional techniques and features. One important change is the inclusion of an in-loop deblocking filter for removal of blocking artifacts. Since the filter can easily account for one-third of the computational complexity of a decoder, its addition was a source of debate during the development of the H.264/AVC standard.

Ample research on architecture design of the deblocking filter has been carried out, generally targeted toward high performance profiles. To the best of our knowledge no other research investigated designs that can be scaled from low-power extended profiles up to high performance profiles.

This work investigated the design of a scalable architecture for the deblocking filter. Four different designs were implemented. The relative performance of the designs were then compared against each other and existing research through simulation. All designs were targeted towards a Xilinx Virtex 5 field programmable gate array (FPGA).

Contents

Dedication . . . . . **iii**

Acknowledgments . . . . . **iv**

Abstract . . . . . **v**

Glossary . . . . . **xi**

1 Introduction **1**
1.1 Thesis Objective 1
1.2 Thesis Chapter Overview 2

2 Background **3**
2.1 Video Basics 4
2.2 Block Based Compression 5
2.3 H.264/AVC Overview 6
2.3.1 Inter-Frame Prediction 7
2.3.2 Intra-Frame Prediction 11
2.3.3 Transformation and Quantization 14
2.3.4 Video Stream Parser 18
2.3.5 H.264/AVC Profiles and Levels 20
2.3.6 H.264/AVC Performance 22
2.4 H.264/AVC Deblocking Filter 23
2.4.1 Deblocking Filter Motivation 23
2.4.2 Deblocking Filter Operation 24

3 Supporting work **32**
3.1 Memory Complexity Management 32
3.2 Filter Complexity 34
3.3 Motivation for presented research 35

4 Design methodology **36**
4.1 Design Components 36
4.1.1 Macroblock-Level Components 36
4.1.2 Frame-Level Components 41
4.2 Implemented Designs 45
4.2.1 Single Serial Filter Design 45
4.2.2 Single Concurrent Filter Design 49
4.2.3 Double Deblocking Filter with Single Edge Filter Design 51
4.2.4 Double Concurrent Filter Design 54

5 Implementation details **55**
5.1 Design component results 55
5.1.1 SR Buffer 55
5.1.2 Matrix Transpose 56
5.1.3 Edge Filter 56
5.1.4 Threshold Derivation 57
5.1.5 Boundary Strength Calculation 57
5.2 Design results 57
5.2.1 Single deblocking filter with serial edge filter 57
5.2.2 Single deblocking filter with concurrent edge filter 60
5.2.3 Double deblocking filter with single edge filter 61
5.2.4 Double deblocking filter with concurrent edge filter 63

6 Testing **65**

7 Analysis of results **68**
7.1 Proposed designs 68
7.2 Comparison to other research 72

8 Conclusions from work **74**

Bibliography **76**

List of Figures

2.1 Video standards history [1] 4
2.2 H.264/AVC primary YCbCr color space - 4:2:0 5
2.3 H.264/AVC encoder 6
2.4 H.264/AVC decoder 7
2.5 Macroblock and submacroblock partitions 8
2.6 Example of integer and sub-sample motion prediction in H.264/AVC 8
2.7 Filtering for fractional-sample accurate motion compensation 9
2.8 Multi-frame motion compensation 11
2.9 H.264/AVC base for 4x4 intra predictions 12
2.10 H.264/AVC 4x4 intra-prediction modes 14
2.11 H.264/AVC 16x16 intrapredction modes 14
2.12 H.264/AVC scanning order 15
2.13 H.264/AVC scanning order 16
2.14 Zig-zag ordering 19
2.15 H.264/AVC primary profile's subsets of features 21
2.16 Filtering over macroblock in H.264/AVC [17] 22
2.17 Filtering over macroblock in H.264/AVC [17] 23
2.18 Filtering over macroblock in H.264/AVC 24
2.19 (a) unfiltered, and (b) filtered 25
2.20 Filtering over macroblock in H.264/AVC 26
2.21 Filtering order for a macroblock in H.264/AVC 27
2.22 Calculation of boundary strength parameters in H.264/AVC 28
2.23 Threshold analysis based on α and β 29

3.1 Filtering orders for macroblock: left - semi-2D from [5] ; right - 2D from [12] 34
3.2 Concurrent Filtering order for macroblock in H.264/AVC - [6] 34

4.1 Interface for the shift register buffer 37
4.2 Interface for the matrix transpose unit 37

4.3 RTL for the matrix transpose unit . . . 37
4.4 Interface for the edge filter . . . 38
4.5 RTL for the edge filter . . . 39
4.6 Interface for macroblock memory . . . 40
4.7 Layout for macroblock memory . . . 40
4.8 Memory map for macroblock memory . . . 40
4.9 Interface for threshold derivation unit . . . 41
4.10 Interface for the boundary strength calculation unit . . . 41
4.11 Needed macroblock samples . . . 42
4.12 Macroblock buffer interface . . . 43
4.13 Macroblock buffer RTL . . . 44
4.14 Interface for the deblocking filter wrapper . . . 44
4.15 Serial edge filter design's data ordering [12] . . . 45
4.16 Single-serial filter design . . . 46
4.17 Top control unit for the serial edge filter wrapper . . . 47
4.18 Load macroblock control unit for the serial edge filter wrapper . . . 48
4.19 Store macroblock control unit for the serial edge filter wrapper . . . 48
4.20 Single-concurrent filter design . . . 49
4.21 Single-conrruent filter design's data ordering [6] . . . 50
4.22 Serial and concurrent frame level filtering of macroblocks . . . 51
4.23 Double-serial filter design . . . 52
4.24 Double top finite state machine . . . 53
4.25 Double-concurrent filter design . . . 54

5.1 Filter output . . . 58
5.2 Filter residual output . . . 59

6.1 Initial setup for testing functionality of deblocking filter - from [16] . . . 66
6.2 Architecture for testing advanced deblocking filter designs . . . 67

7.1 Speedup in filtering time relative to single-serial design . . . 69
7.2 Speedup in total filtering time, including reading and writing, relative to single-serial design . . . 70
7.3 Speedup in total filtering time, including reading and writing, relative to single-serial design . . . 71
7.4 Speedup for each design divided by gate count . . . 72

List of Tables

2.1 Intra 4x4 prediction modes [10] 13
2.2 Inta 16x16 prediction modes [10] 13
2.3 Multiplication factor MF for $0 \leq QP \leq 5$ [10] 17
2.4 NAL unit format 20

7.1 Performance relative to published research 73

Glossary

1080P Video resolution with image size 1920x1080 pixels, p. 1.

CABAC Context Adaptive Binary Arithmetic Coding. An efficient entropy coding standard used in the H.264/AVC Main and High Performance Profiles, p. 19.

CAVLC Context Adaptive Variable Length Coding. An improved variable length coding entropy used in H.264/AVC, p. 19.

DCT Discrete Cosine Transform, p. 14.

FIR Finite Impulse Response, p. 8.

FRExt Fidelity Range Extensions - amendment to the H.264/AVC standard defining an additional 4 high fidelity profiles, p. 20.

H.264/AVC Video coding standard approved in spring 2003 by both ISO/IEC and ITU-T., p. 3.

ISO/IEC International Organization for Standardization and International Engineering Consortium, p. 3.

JVT Joint Venture Team, p. 3.

MPEG Motion Pictures Expert Group, p. 3.

NAL Network Abstraction Layer, p. 20.

NTSC National Television Systems Committee, p. 4.

PSNR Peak Signal-to-Noise Ratio, p. 22.

RGB Red Green Blue color scheme, p. 4.

SQCIF Sub Quarter Common Intermediate Format, p. 4.

VCEG Video Coding Experts Group, p. 3.

VCL Video Coding Layer, p. 20.

YCbCr Color format - Y is luma, Cb is chroma blue, and Cr is chroma red, p. 4.

Chapter 1

Introduction

1.1 Thesis Objective

Because of its improvements in bit-rate savings and perceptual quality, H.264/AVC is receiving widespread adoption. One of its major improvements, the addition of an in-loop deblocking filter, significantly increases the complexity of the decoder design. For this reason, there has been ample research seeking efficient and effective designs for deblocking filters. This thesis provides an in depth look at the design of a scalable architecture for the H.264/AVC deblocking filter.

It is important to investigate ways to reduce the deblocking filter complexities and to create efficient architectures for meeting performance requirements for a range of target products. The design specifications will vary greatly depending on the targeted application. For an ultra-mobile product, such as an Apple iPhone, the least complex and lowest power codec that will meet the video requirements will be sought. For the latest 1080P performance display, the codec will be much larger, where power is neglected in lieu of performance. This research approach was unique in that it developed a simple design, and then iteratively scaled the complexity to target higher performance profiles. Doing this allowed examination of scaling techniques that can be applied for targeting different video formats. Four different designs were implemented, each with increasing complexity but similar base components. The relative performance of the designs were then compared against each other and existing research through post-synthesis simulation of video test

sequences. All designs were targeted towards a Xilinx V5-LX85 FPGA.

1.2 Thesis Chapter Overview

This thesis document starts with an overview of the development of the H.264/AVC standard in chapter 2. Next an introduction to video basics is presented, followed by a summary of the H.264/AVC standard. After an understanding of the standard is given, a more detailed description of the deblocking filter is provided. Chapter 3 presents past research on deblocking filter designs and an analysis on why the proposed research is unique and necessary. The different design elements and the investigated implementations are described in chapter 4. The implementation process and results for the designs are given in chapter 5. The testing environment, and varying methods of testing used are detailed in chapter 6. Finally, chapter 7 concludes this thesis with a discussion of the results, and a look into possible future work and improvements.

Chapter 2

Background

Two main groups responsible for standardization of video compression techniques are the Video Coding Experts Group (VCEG) of the Telecommunications Standardization Sector of the International Telecommunications Union (ITU-T) and the Moving Picture Experts Group (MPEG) of the International Organization for Standardization and International Engineering Consortium (ISO/IEC). Each organization develops standards targeted towards applications ranging from handheld wireless to high definition television broadcasts. VCEG generally develops standards targeted towards low bit-rate communication, while MPEG focuses on high performance standards.

In the past, VCEG and MPEG have come out with a number of major standards, as seen in figure 2.1. The most notable of these is the first joint venture between MPEG and VCEG, H.262/MPEG-2. This standard is widely used for television broadcast (SDTV and HDTV) and is usually used as a DVD codec. Another major development by VCEG was the H.263 standard, generally targeting low bit-rate compressed formats. MPEG's MPEG-4 Part 2 ASP is based off of H.263, with the addition of some advanced features [17].

MPEG and VCEG formed a Joint Video Team (JVT) in 2001 to expedite the development of a new standard. The joint venture, submitted in March of 2003, is referenced by many names: H.264, H.26L, ISO/IEC 14496-10, H.264/AVC, MPEG-4 AVC and MPEG-4 Part 10. For clarity, the standard will be called H.264/AVC for the remainder of this document. The resulting standard represents the single largest improvement in coding efficiency and perceptual quality since MPEG-2, and it is expected that it will replace other standards

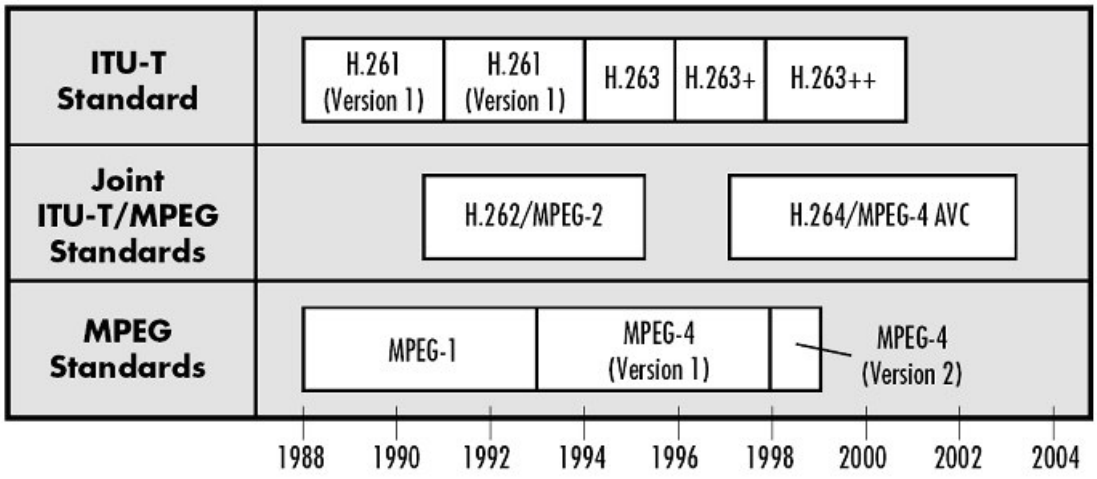

Figure 2.1: Video standards history [1]

over time [1].

2.1 Video Basics

Video signals can be viewed as series of pictures that are displayed over time. The number of pictures displayed per second, or frame rate, determines how smooth changes in the scene will be perceived. Frame rates of 12 or 15 are typical for low bit-rate video using Sub Quarter Common Intermediate Format (SQCIF), while 24 or 30 is standard for National Television Systems Committee (NTSC) television. Many new high performance video distributors propose using up to 60 frames per second on 1080p (1920x1080 image dimensions) content, to provide even more smooth apparent motion [10], [3].

Color spaces are used to digitally encode pixels within frames. RGB, a common and simple color space, divides pixels into three components, R (red), G (green) and B (blue). The YCbCr color space is used in H.264/AVC, and is also composed of three components. The luma component, Y, represents brightness (weighted sum of non-linear R'G'B' components after gamma correction), while the two chroma components, Cb and Cr, represent how much a color deviates from gray toward blue or red, respectively. The human visual

system perceives scenes based on brightness and color content separately, with a greater sensitivity to changes in brightness detail over color [18]. In H.264/AVC, YCbCr can take advantage of this by representing color (chroma) components using fewer samples relative to brightness (luma) components. This can be seen in figure 2.2, where a 4:2:0 color space is used (chroma components with half the number of bits compared to luma).

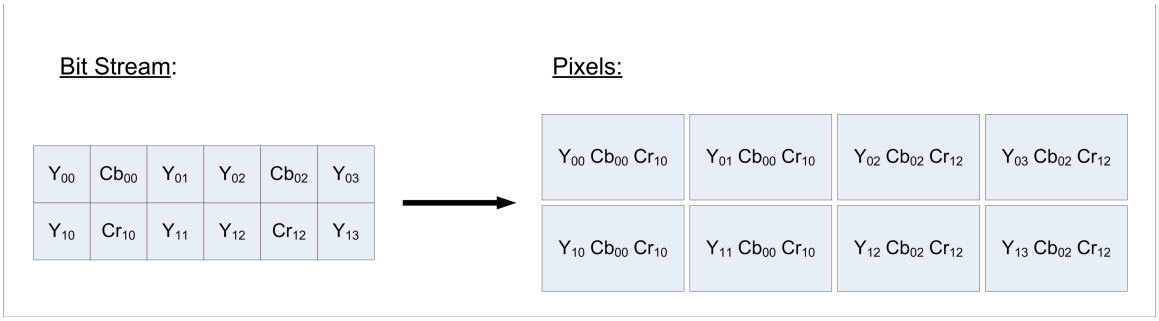

Figure 2.2: H.264/AVC primary YCbCr color space - 4:2:0

2.2 Block Based Compression

In block-based compression standards, such as H.264/AVC, each video frame is divided into macroblocks, which are the basic image elements on which encoding and decoding is specified. A macroblock is a rectangular area consisting of 16x16 luma samples and 8x8 samples of both chroma components. For example, in standard definition video (720x480), each frame is divided into 1,350 (45x30) macroblocks. This definition of a macroblock has been used in all previous VCEG and MPEG coding standards.

A frame is coded as one or more slices, each containing one or more macroblocks. In a given picture, it is possible to have multiple slice groups. A slice group defines particular decoding parameters for a given set of macroblocks. Each slice should be coded independently of other slices in a given frame.

2.3 H.264/AVC Overview

The high level diagram for the encoding process in H.264/AVC, as described in [10], is shown in Figure 2.3.

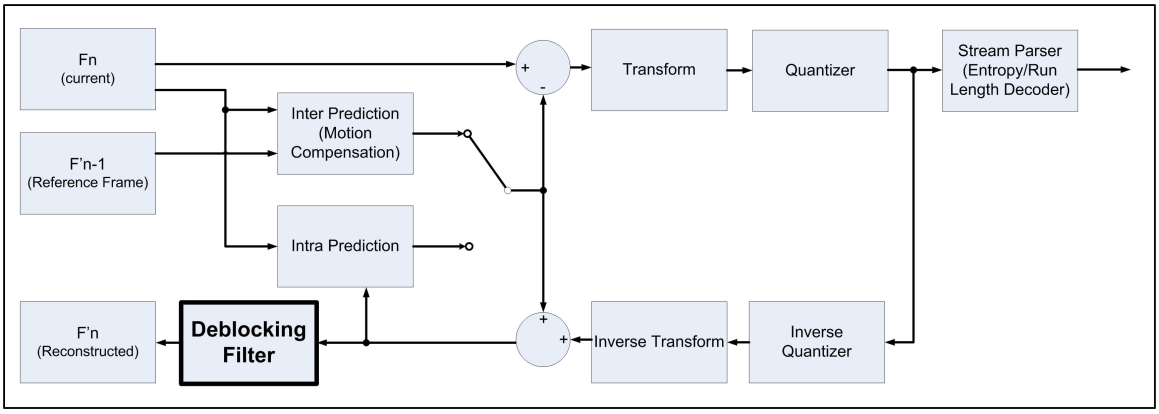

Figure 2.3: H.264/AVC encoder

Uncompressed video flows through the motion compensation or intra-prediction blocks, transform and then the quantization stages. The data is then fed back through inverse quantization and inverse transform stages. These pseudo-decoder feedback stages provide data that is used to calculate residual values - the difference between the actual incoming video and the estimated, transformed and quantized frames that would be seen by a decoder. The residuals are then encoded along with settings and parameters used for motion compensation, transformation and quantization to estimate the original frame. The data is then sent through entropy coding. Until it reaches the run length decoder, the data-flow operates on macroblocks of data, which allows pipelined processing for streaming video. The encoder parameters are sent in macroblock or slice headers. The run length decoder takes samples and header information and forms a packet of data for compressed transmission (or storage). The decoder block diagram can be seen in Figure 2.4, which behaves in the same manner as the feed-back path in the encoder. Following a more thorough description of the stages, an overview of H.264/AVC's performance and profiles is given.

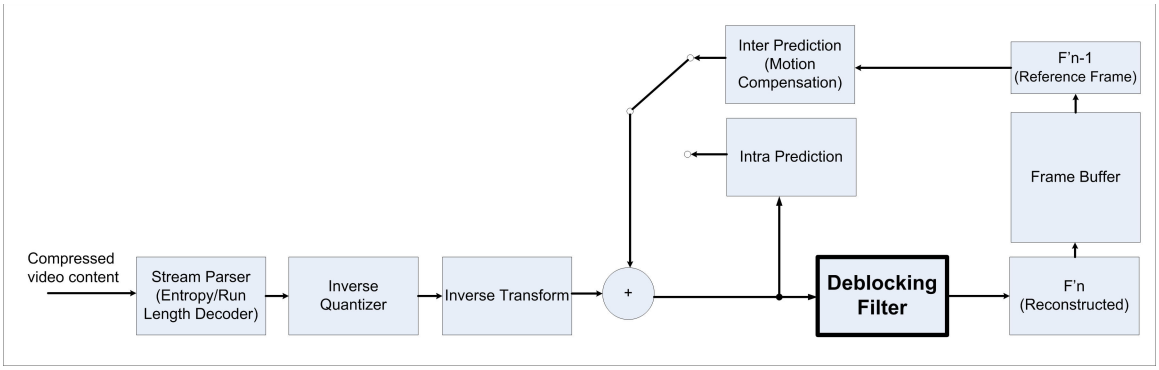

Figure 2.4: H.264/AVC decoder

2.3.1 Inter-Frame Prediction

As a result of multiple frames being displayed every second, consecutive images may contain similar data. Motion estimation examines sequences of image frames, looking for temporal redundancies. Inter-prediction creates a prediction model for an estimated block using blocks from previously encoded video frames. Motion estimation is improved in H.264/AVC over previous standards by including support for many different block sizes, which in turn may be composed of a combination of various sub-blocks. This method of partitioning marcoblocks into motion blocks and sub-blocks is known as tree structure motion compensation. The defined macroblock partitions in H.264/AVC can be seen in Figure 2.5.

Every inter-predicted sub-block is predicted based on a sub-block of equal dimensions from a previously encoded reference image. The offset in location between the reference and predicted block is called a motion vector. It is possible that the motion vector will not be an integer number of samples. As a result, interpolation must be used to calculate non-integer motion vectors, and a finite vector resolution will exist. For H.264/AVC, luma components have quarter-sample resolution and chroma components have one-eighth-sample resolution. An example of integer and sub-sample predictions can be seen in Figure 2.6. Figure 2.6 (b) shows an integer reference motion vector from (a), while (c) shows a sub-sample motion vector.

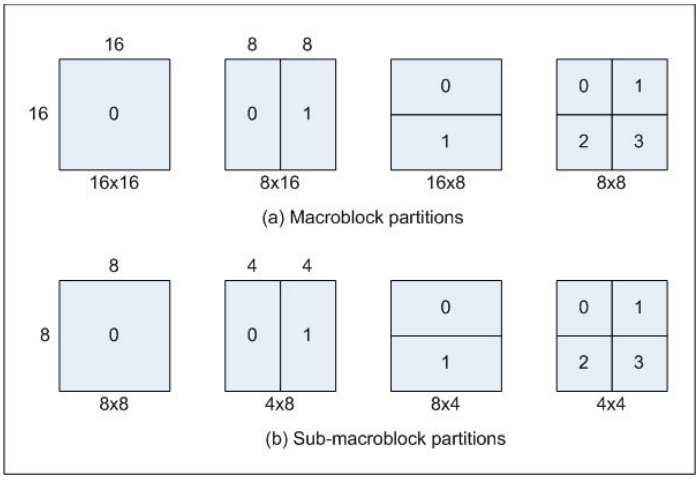

Figure 2.5: Macroblock and submacroblock partitions

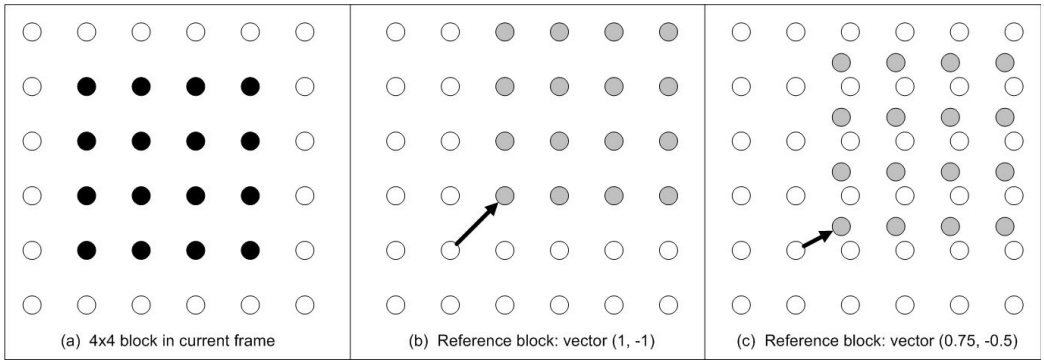

Figure 2.6: Example of integer and sub-sample motion prediction in H.264/AVC

The prediction values for one-half sample positions are calculated by applying a 6-tap FIR low pass filter horizontally and vertically. Quarter-sample positions are obtained by taking the mean of the samples at integer- and half-sample positions. The process of estimating sub-sample accurate motion compensation is depicted in figure 2.7. The uppercase letters (A,B,...U) indicate actual pixels in the reference frame. The lower case letters represent sub sample (half or quarter) locations. Half-sample locations are generated first, using the 6-tap filter. In 2.7, b_1 is an unclipped half-sample location, and is obtained by filtering the horizontal row of pixels. Similarly, the vertical column of pixels are used to derive h_1.

$$b_1 = (E - 5F + 20G + 20H - 5I + J) \tag{2.1}$$

$$h_1 = (A - 5C + 20G + 20M - 5R + T) \tag{2.2}$$

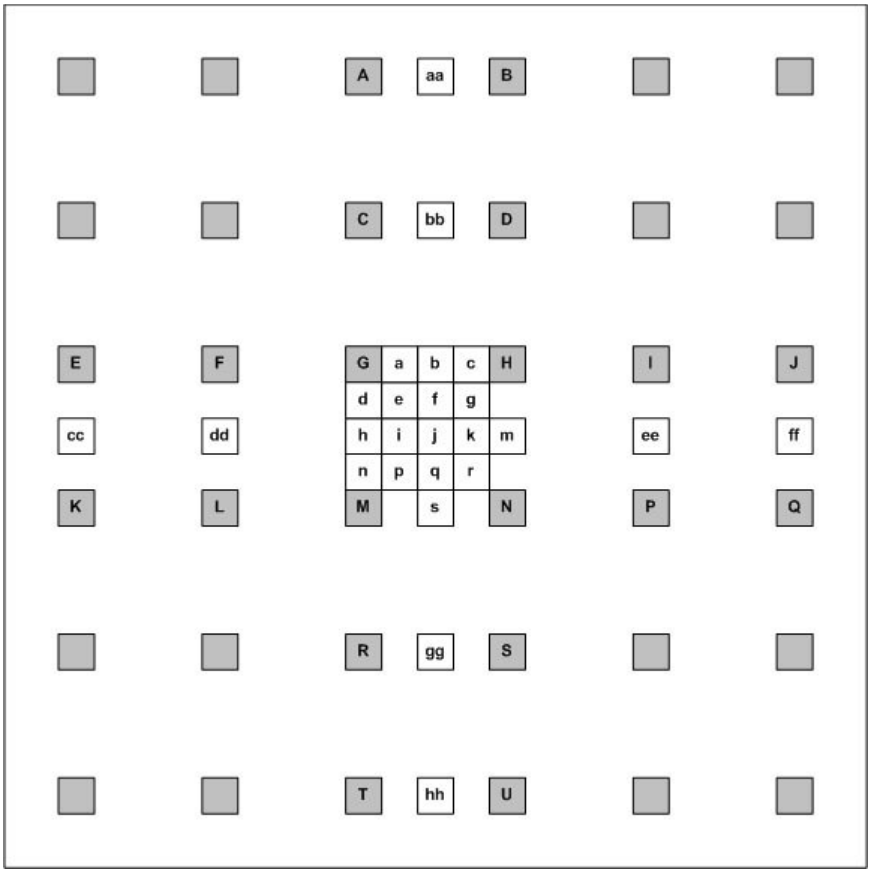

Figure 2.7: Filtering for fractional-sample accurate motion compensation

These samples are then rounded as follows, and then clipped to the range of 0 to 255, to provide the half-sample positions b and h.

$$b = (b_1 + 16) \gg 5 \tag{2.3}$$

$$h = (h_1 + 16) \gg 5 \tag{2.4}$$

Half-pixel position j is calculated by interpolating other half-pixel samples (cc, dd, ee and ff: calculated similarly as h).

$$j = ((cc - 5dd + 20h_1 + 20m_1 - 5ee + ff) + 512) \gg 10 \tag{2.5}$$

The quarter-samples at a, c, d, n, f, i, k and q are derived by taking the average of the two nearest samples at integer and half sample positions. For example:

$$a = (G + b + 1) \gg 1 \tag{2.6}$$

The quarter samples at e, g, p and ar are calculated by taking the mean of the two nearest half-samples in the diagonal direction. For example:

$$e = (b + h + 1) \gg 1 \tag{2.7}$$

Chroma components are calculated in a similar manner as luma, only with a one-eighth sample resolution.

Since each sub-partition of a motion block requires its own motion vector, calculating and sending every motion vector would become costly. There is a high correlation between motion vectors of a macroblock and neighboring macroblocks. H.264 takes advantage of this by predicting motion vectors based on neighboring partitions' vectors to save on the amount of data needed to be sent. Based on this, only the differences between the predicted motion vector (which can be encoded using fewer bits) and the actual motion vector (the residual) is sent in the data stream.

H.264/AVC provides a major improvement over previous standards by facilitating multiple reference frames. As noted in [1], this technique is useful for:

- Motion that is periodic in nature
- Translating motion and occlusions
- Alternating camera angles that switch back and forth

An example of multiple frame motion estimation can be seen in Figure 2.8.

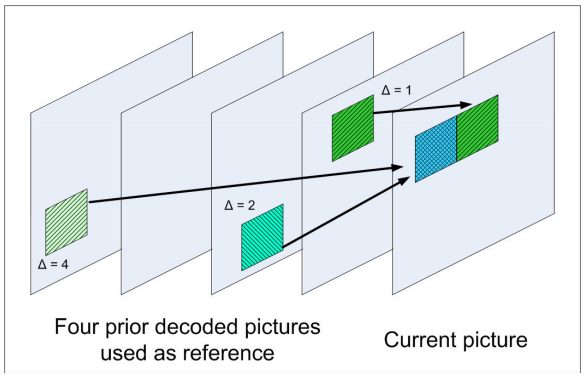

Figure 2.8: Multi-frame motion compensation

To further enhance its motion estimation capabilities, referencing was decoupled from temporal order. In previous standards, reference order and display order were one in the same. With this restriction removed, the encoder is allowed to choose the order of pictures for reference purposes irrespective of the display ordering in order to optimize the decode process.

Three types of picture frames are distinguished for motion compensation blocks: I-frame, P-frame, B-frame. An I-frame is coded independently of any other frame, and acts as a baseline reference for other frames to be encoded. These result in the least amount of compression. P-frames are coded predictively with reference to previous frames. B-frames are bi-directionally predictive-coded frames, coded similar to P-frames, except prediction coding is based upon both past and future frames. These result in the greatest amount of compression [18].

2.3.2 Intra-Frame Prediction

Intra-frame prediction is another feature added in the H.264/AVC standard and is used to reduce spatial redundancies. This unit operates in the spatial domain (after the inverse transform block). Intra estimation predicts pixel values by extrapolating neighboring pixels from adjacent reconstructed blocks. Three different intra-coding modes are available:

- 4x4 intra prediction: Predicts each 4x4 luma block based on bordering 4x4 blocks

and nine different directions. This mode is used when coding image portions with a significant level of detail.

- 16x16 intra prediction: Predicts the luma portion of each macroblock in a single operation based on neighboring macroblocks. This mode is used when coding smooth portions of an image.

- I_PCM: Transform and prediction stages are bypassed, allowing for the transmission of image samples directly. This can be a more efficient operation mode when operating with very low quantizer parameters.

Intra 4x4 Prediction Mode

A 4x4 luma block that is required to be predicted is shown in Figure 2.9. The samples above and to the left (labeled A-M) are available as reference for encoding the luma block. The samples a-p in the figure are calculated based on A-M in one of the modes described in Table 2.1. Figure 2.10 shows the various modes of operation.

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

Figure 2.9: H.264/AVC base for 4x4 intra predictions

Intra 16x16 Prediction Mode

The 16x16 intra-prediction mode predicts all of the luma components of a macroblock in a single operation. This prediction is faster and is useful in smooth regions of the frame. This prediction has four modes of operation, as described in table 2.2: vertical, horizontal, DC, and plane prediction. These prediction modes can be seen in Figure 2.11.

Mode 0 (vertical)	The upper samples A, B, C, D are extrapolated vertically.
Mode 1 (horizontal)	The left samples I, J, K, L are extrapolated horizontally.
Mode 2 (DC)	All samples are predicted by the mean of samples A...D and I...L.
Mode 3 (Diagonal down-left)	Samples are interpolated at a 45° angle between lower-left and upper-right.
Mode 4 (Diagonal down-right)	Samples are extrapolated at a 45° angle between down and to the right.
Mode 5 (Vertical-right)	Samples are extrapolated at a 26.6° angle to the left of vertical.
Mode 6 (Horizontal-down)	Samples are extrapolated at a 26.6° angle below horizontal.
Mode 7 (Vertical-left)	Samples are extrapolated (or interpolated) at a 26.6° angle to the right of vertical.
Mode 8 (Horizontal-up)	Samples are interpolated at a 26.6° angle above horizontal.

Table 2.1: Intra 4x4 prediction modes [10]

Mode 0 (vertical)	Extrapolation from upper samples (H).
Mode 1 (horizontal)	Extrapolation from left samples (V).
Mode 2 (DC)	Mean of upper and left-hand samples (H,V).
Mode 3 (Plane)	linear plane function fitted to the upper and left-hand samples (H,V).

Table 2.2: Inta 16x16 prediction modes [10]

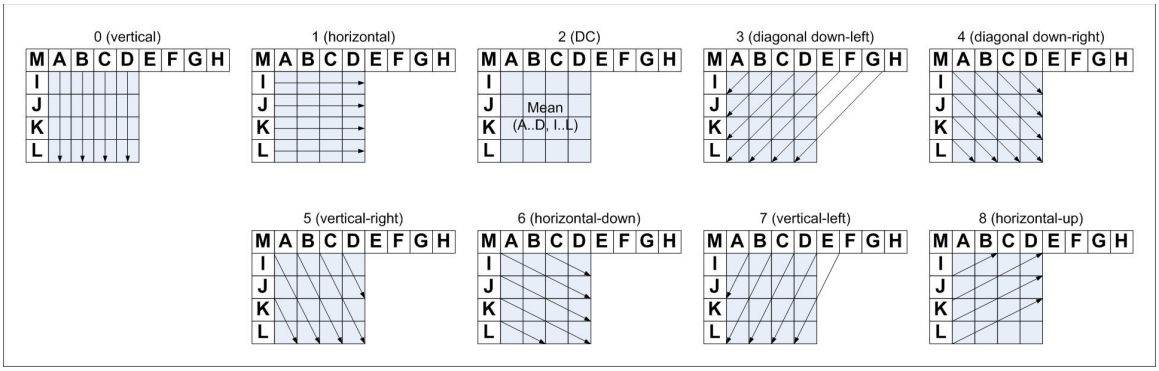

Figure 2.10: H.264/AVC 4x4 intra-prediction modes

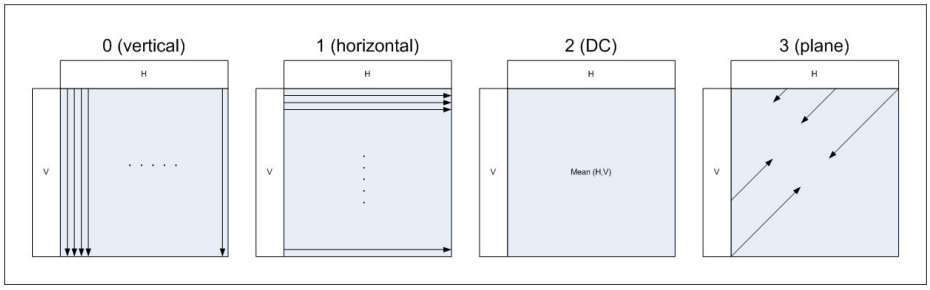

Figure 2.11: H.264/AVC 16x16 intrapredction modes

Chroma Prediction Mode

Each 8x8 chroma component is intra coded similarly to the luma 16x16 prediction mode. The block is predicted using reconstructed chroma samples from above and to the left. The prediction is carried out using the same four modes described in figure 2.11.

2.3.3 Transformation and Quantization

An integer transform which approximates the 2D discrete cosine transform (DCT) is used in the transform stage. This stage converts image data from the spatial to the frequency domain. The quantization stage is next used to remove a portion of the higher frequency coefficients created in the transform stage. There is a trade off between the number of coefficients removed (amount of compression) and perceptual quality. At the output of the quantization stage, data is ordered from low to high frequencies, with high tending to be

zero, resulting in a data pattern that is very suitable for compression by the following stage.

Macroblock data is transmitted to the transform stage in the order shown in Figure 2.12. If the macroblock is coded using 16x16 intra prediction, then the block labeled -1, the DC coefficients, is sent first. The luma residual blocks 0-15 are transmitted next in the order shown. Blocks 16 and 17 are then sent, creating a 2x2 array of DC coefficients from the Cb and Cr chroma components. Finally, chroma residual blocks 18-25 are sent. Depending on the data being transmitted, one of three different types of transforms may be

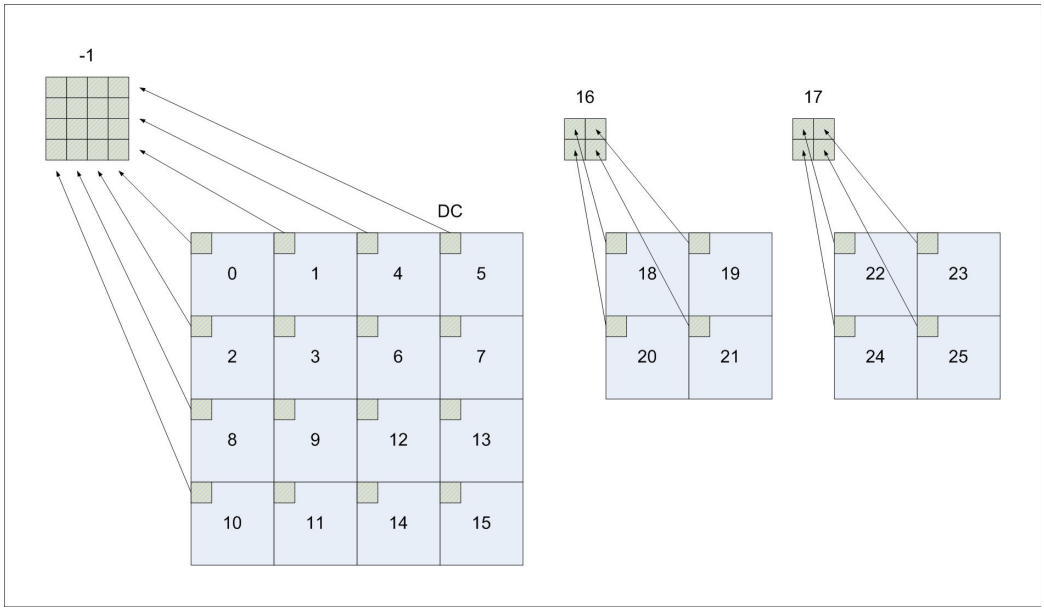

Figure 2.12: H.264/AVC scanning order

used: a Hadamard transform for the 4x4 array of luma DC coefficients in intra macroblocks predicted using 16x16 mode; a Hadamard transform for the 2x2 array of chroma DC coefficients; and a DCT-based transform for all 4x4 blocks of residual data. All 4x4 residual blocks first pass through the DCT-based transform, while DC coefficients may undergo a second transformation depending on the prediction type used. This transformation process can be seen in 2.13.

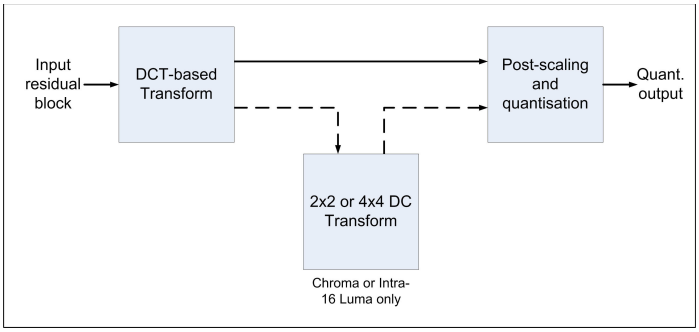

Figure 2.13: H.264/AVC scanning order

4x4 residual transform and quantization

This DCT-based transform is an integer transform, allowing all operations to be carried out without a loss of accuracy that is inherent in floating point operations. The core part of the transform can be implemented using only additions and shifts. This allows for reduced design complexity since no multiplications are needed. Also, the scaling multiplication portion of the transform can be integrated into the quantizer, reducing the total number of operations needed. The transform is defined as follows:

$$a = \frac{1}{2}, \quad b = \sqrt{\frac{2}{5}} \tag{2.8}$$

$$Y = C_f\,\mathbf{X}\,C_f^T \otimes E_f = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} \\ \mathbf{X} \\ \\ \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \tag{2.9}$$

The CXC^T portion of the transform is the core of the 2D transform, while the E_f matrix is a scaling factor that will be combined with the scaling in the quantization stage, allowing for a one less multiplication. The transformed coefficients without the scaling matrix will be denoted as $W_{(i,j)}$. For all AC coefficients, quantization will carried out next.

QP	Positions (0,0),(2,0),(2,2),(0,2)	Positions (1,1),(1,3),(3,1),(3,3)	Other positions
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

Table 2.3: Multiplication factor MF for $0 \leq QP \leq 5$ [10]

The amount of quantization used depends on the quantization parameter, QP. Shown below, the $Y_{D(i,j)}$ coefficients are quantized to produce the output coefficients $Z_{D(i,j)}$:

$$\left|Z_{D(i,j)}\right| = \left(\left|W_{(i,j)}\right| MF + f\right) \gg qbits \tag{2.10}$$

where

$$qbits = 15 + floor(QP/6) \tag{2.11}$$

and

$$f = \begin{cases} 2^{qbits}/3 & : \quad Intra - blocks \\ 2^{qbits}/6 & : \quad Inter - blocks \end{cases} \tag{2.12}$$

The value of the multiplying factor, MF, is determined based on table 2.3.

4x4 Luma DC Coefficient Transform and Quantization for 16x16 Intra-mode

After executing the previously discussed core 4x4 transform on the luma DC coefficient, a 4x4 Hadamard transform is additionally used to transform the coefficients when using 16x16 intra-prediction mode. This transformation is shown below:

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \\ W_D \\ \\ \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) / 2 \tag{2.13}$$

Following this transformation, quantization is carried out on the coefficients.

$$\left|Z_{D(i,j)}\right| = \left(\left|Y_{D(i,j)}\right| MF_{(0,0)} + 2f\right) \gg (qbits + 1) \tag{2.14}$$

2x2 Chroma DC Coefficient Inverse Transform and Quantization

After the Cr and Cb DC Chroma coefficients are transformed using the previously discussed core 4x4 transform, a 4x4 Hadamard transform is applied.

$$W_{QD} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} W_D \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.15}$$

After the transformations, quantization is carried out on the coefficients as shown below:

$$\left|Z_{D(i,j)}\right| = \left(\left|Y_{D(i,j)}\right| MF_{(0,0)} + 2f\right) \gg (qbits + 1) \tag{2.16}$$

2.3.4 Video Stream Parser

A H.264/AVC video stream is composed of video packets. The video stream parser takes 4x4 scaled coefficients from the quantizer and applies entropy coding, finally organizing the data into video packets for efficient transmission or storage.

Data Reordering

Entropy coding is carried out on a serial bit stream, so the 4x4 coefficients must be serialized after quantization. Data reordering is applied before entropy coding in order to provide

coefficients in as an efficient order as possible. The ordering shown in figure 2.14 reorders the quantized coefficients from low frequency to high frequency.

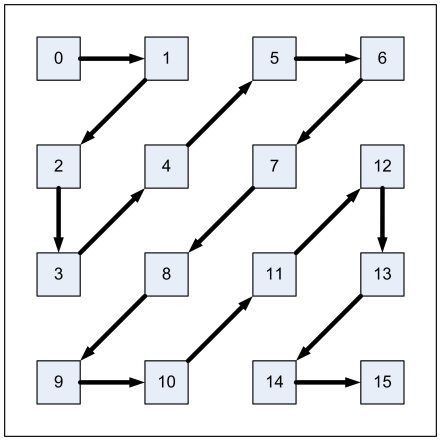

Figure 2.14: Zig-zag ordering

Entropy coding

Entropy coding is a lossless technique which compresses the final serial data stream by mapping frequently used symbols to short bit codes, while mapping less frequently used symbols to larger bit codes. Three different types of entropy codings are used in H.264/AVC: basic, Exp-Golomb and context adaptive coding. Syntax elements above the slice layer are encoded as fixed or variable length binary codes. At or below the slice layer, variable length or Context Adaptive Arithmetic Coding (CABAC) is used to encode elements. When using variable length, residual block data is coded using Context Adaptive Variable Length Coding (CAVLC) and other syntax elements are coded using Exp-Golomb schemes. When CABAC is used, both residual and syntax elements are coded using the arithmetic-coding scheme. CABAC offers superior coding over CAVLC at the expense of complexity by: adapting to changing probability distribution of symbols; exploiting correlations between symbols; and by adaptively exploiting bit correlations using arithmetic coding [cite white paper].

Field	Description
forbidden_zero_bit	1-bit fixed (='0')
nal_ref_idc	2-bit unsigned number
nal_unit_type	5-bit unsigned number
rbsp_byte[]	Rest of bytes in NAL unit (the Raw Bit Sequence Payload (RBSP))

Table 2.4: NAL unit format

Network abstraction layer parsing

H.264/AVC defines a network abstraction layer (NAL) for efficient transmission of video data and header information. A coded H.264 video sequence consists of a series of network abstraction layer units, each unit containing video data or header information. There are two main types of NAL units: video coding layer (VCL) and non-VCL units. VCL units contain video sample data while non-VCL units contain header information (which may be associated with multiple VCL units). Each NAL unit is organized as shown in Table 2.4.

2.3.5 H.264/AVC Profiles and Levels

The H.264/AVC standard originally defined three primary profiles for decoders and encoders, targeting different video applications: Baseline, Main and Extended. Each profile defines a subset of features supported by all codecs conforming to that profile. The Baseline profile generally targets real-time conversation services such as networked video conferencing, providing minimal complexity with high robustness and flexibility. The Main profile is designed for digital storage media and television broadcasting. The Extended profile is aimed at multimedia services over the internet, combining the robustness from the Baseline profile with increased coding efficiencies [15]. Figure 2.15 shows the subset of features and overlap between the primary three profiles.

Additional high performance profiles have been defined in the fidelity range extensions (FRExt), targeting content-distribution and studio editing and post-processing. The FRExt

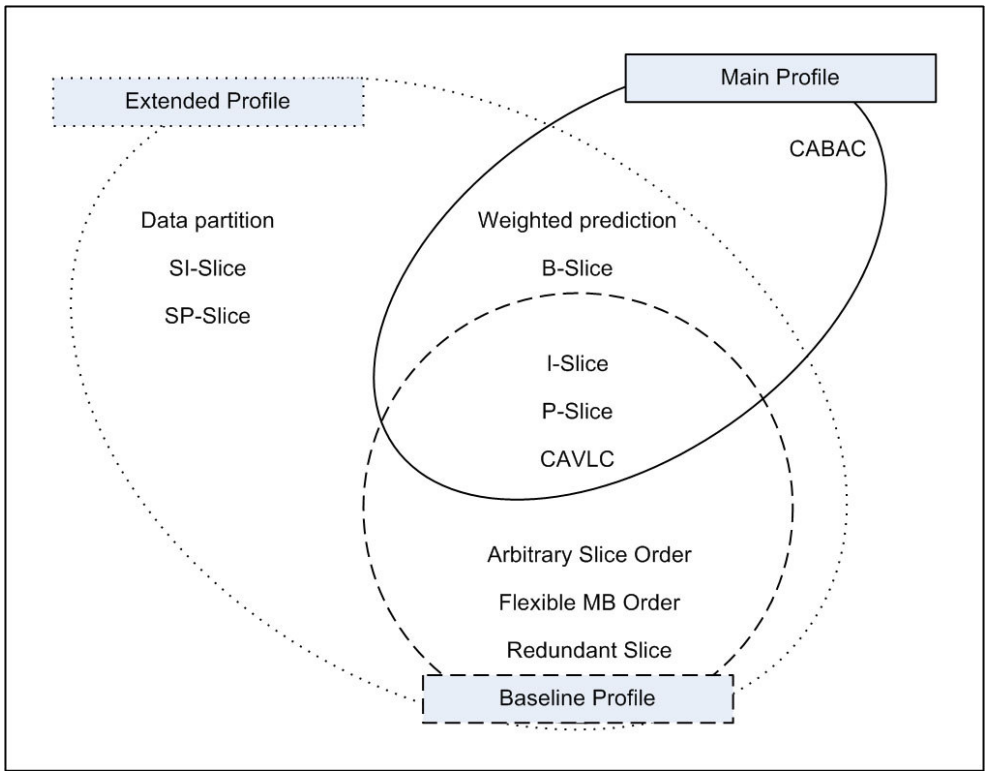

Figure 2.15: H.264/AVC primary profile's subsets of features

defines four additional profiles, providing different higher video resolutions through various features, including [15]:

- Higher sample accuracy - from 8 bits to 12 bits per sample
- Higher color accuracy - 4:2:0, 4:2:2 and 4:4:4 chroma sampling
- Support for efficient lossless representation

Beyond profiles, the standard also defines levels for codecs. For any given profile, levels correspond to the processing and memory capabilities of the codec. For example, levels can define the necessary frame-rate, image resolution and transmission bit-rate for a corresponding codec.

2.3.6 H.264/AVC Performance

For a variety of video types, [17] measured the peak signal-to-noise ratio (PSNR) for given bit-rates on sequences encoded and decoded by a number of standards. The codec with the lowest PSNR for given bit-rates has the best performance. Figure 2.16 shows the resulting PSNR against bit-rate for MPEG2, H.263, MPEG-4-ASP and H.264/AVC (main profile) on the QCIF Foreman sequence. For a given bit-rate, H.264/AVC has the highest signal-to-noise ratio.

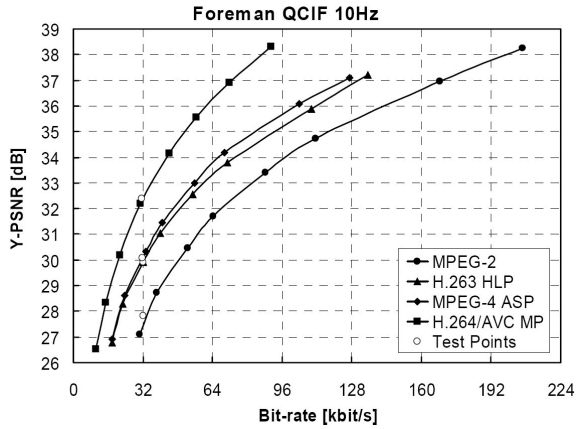

Figure 2.16: Filtering over macroblock in H.264/AVC [17]

For the same video sequence, [17] shows the amount of bit-rate savings possible for a given PSNR relative to MPEG-2 in figure 2.17. Again, H.264/AVC outperforms the rest, providing the greatest bit-rate savings for a given signal-to-noise ratio.

On entertainment workloads, [17] measured an average 45% saving for H.264/AVC over MPEG-2 for a given PSNR. The given results shows how important H.264/AVC will be in the future as a leading standard for a wide range of video applications.

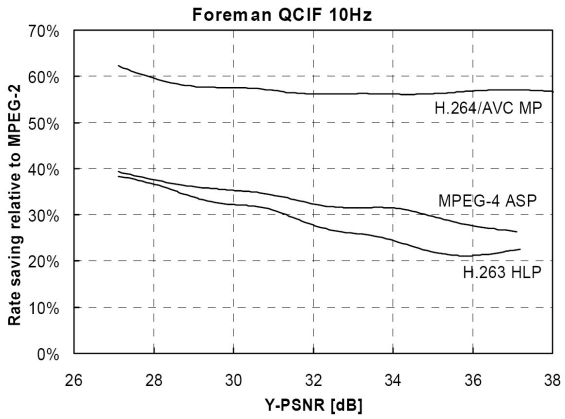

Figure 2.17: Filtering over macroblock in H.264/AVC [17]

2.4 H.264/AVC Deblocking Filter

Blocking artifacts are an inherent issue with block-based video coding schemes. In H.264/AVC, the most significant source of these artifacts is coarse quantization of coefficients from the integer transform used in the inter frame prediction error coding [8]. Blocking artifacts are also sourced from imperfect motion compensation prediction.

2.4.1 Deblocking Filter Motivation

The deblocking filter was included in the H.264/AVC decoder to deliver a better level of perceptual quality in the decode process by removing blocking artifacts. In previous standards, a deblocking filter was optional, and would usually be used for post filtering. The placement of the deblocking filter in H.264/AVC is on the inner-loop, as seen in Figure 2.4.

An image, seen in figure 2.18, is encoded and decoded using H.264/AVC without a deblocking filter. Blocking artifacts can clearly be seen in the result, shown in figure 2.19.

Figure 2.19 shows the result with the deblocking filter. The PSNRR of the unfiltered image is 82.68 dB, while the PSNRR for the filtered image is 84.36 dB. This single example shows the importance of the deblocking filter in the video standard.

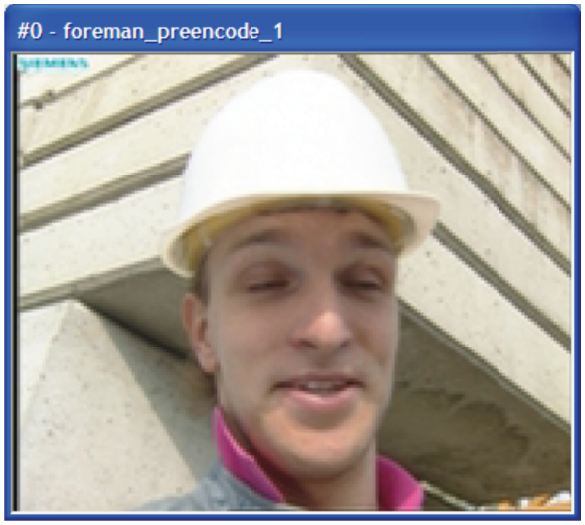

Figure 2.18: Filtering over macroblock in H.264/AVC

2.4.2 Deblocking Filter Operation

Inputs to the deblocking filter include macroblock pixels, boundary strength and threshold values. Filtering is carried out over the vertical edges and then horizontal edges of both the luma and chroma components of the macroblock. Figure 2.20 illustrates the filtering process. Vertical edge 0 of the luma component is filtered first using horizontal filtering from the top of the edge to the bottom (top of figure), followed by edges 1, 2 and 3. The edge filter takes as its input eight pixels: p3, p2, p1, p0, q0, q1, q2, q3. Based on these pixel values, and other threshold values, the shaded pixels (p2, p1, p0, q0, q1, q2) may be adjusted.

Vertical filtering is carried out similarly after horizontal filtering has completed. Because of the overlap in filtering between edges, and between vertical and horizontal filters, it is possible that pixels could be adjusted up to four times. Chroma samples are filtered in the same manner as luma, except only 5 pixels are used: p1, p0, q0, q1, q2, and only p0 and q0 may change. The basic filtering order, as defined for H.264/AVC, is shown in Figure 2.21

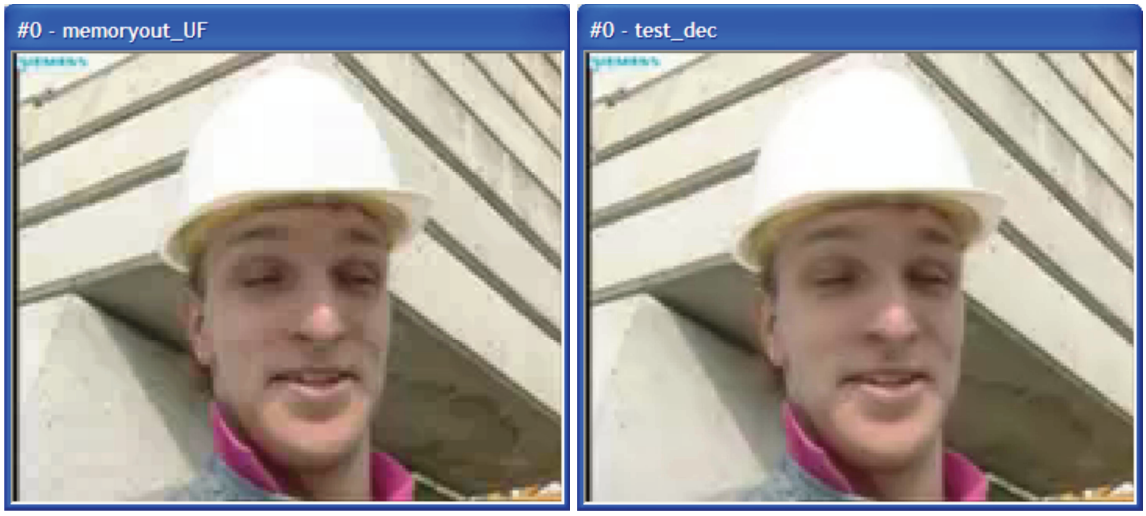

Figure 2.19: (a) unfiltered, and (b) filtered

Determination of Filtering Parameters

The deblocking filter algorithm is highly adaptive, working at three different levels: slice level, block edge level, and sample level. On the block edge level, filtering depends on inter/intra predictions, differences in motion vectors and coded residuals. It is within the block edge level that the strength of filtering is determined by the boundary strength, Bs. Determination of boundary strength is shown Figure 2.22. When filtering is performed, Bs determines the strength of filtering to be carried out. A Bs of 0, for example, results in no filtering, while a Bs of 4 results in the strongest filtering. A summary of filtering for given boundary strengths is:

(a) Bs = 0: No filtering

(b) Bs $\in$ [1,2,3]: A 4-tap linear filter is applied, producing P0 and Q0, and possible P1 and Q1.

(c) Bs = 4: A 3-tap, 4-tap or 5-tap filter may be applied to produce new values of P0, P1, P2, Q0, Q1 and Q2, depending on threshold values and the actual pixel values.

On the sample level, the deblocking filter attempts to distinguish between true edges and those created by blocking artifacts. Through the values of the quantization parameters (QP) α and β, the samples across an edge boundary are analyzed as shown in Figure 2.23.

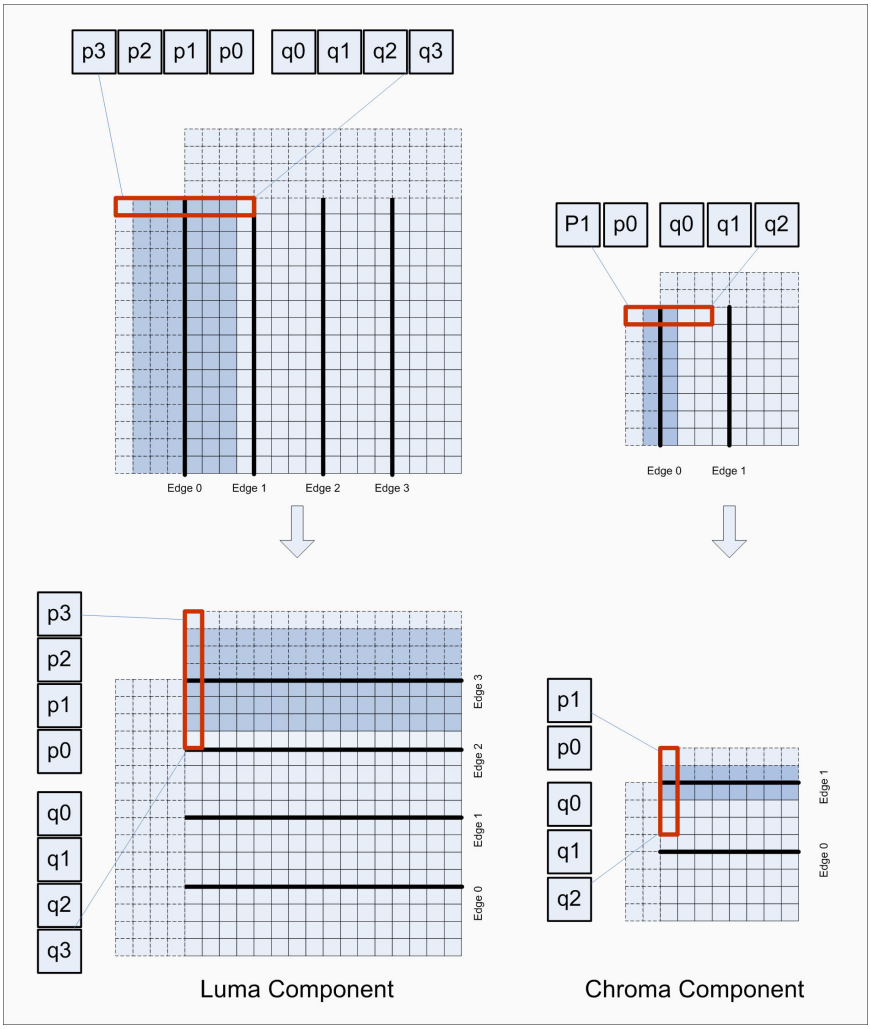

Figure 2.20: Filtering over macroblock in H.264/AVC

If Bs is greater than zero, filtering only takes place if the following three conditions are met:

$$|p_0 - q_0| < \alpha(Index_A) \tag{2.17}$$

$$|p_1 - p_0| < \beta(Index_B) \tag{2.18}$$

$$|q_1 - q_0| < \beta(Index_B) \tag{2.19}$$

In the above equations, thresholds α and β are dependent on both the average quantization parameter (QP) and encoder defined offsets, $Offset_A$ and $Offset_B$, as shown in

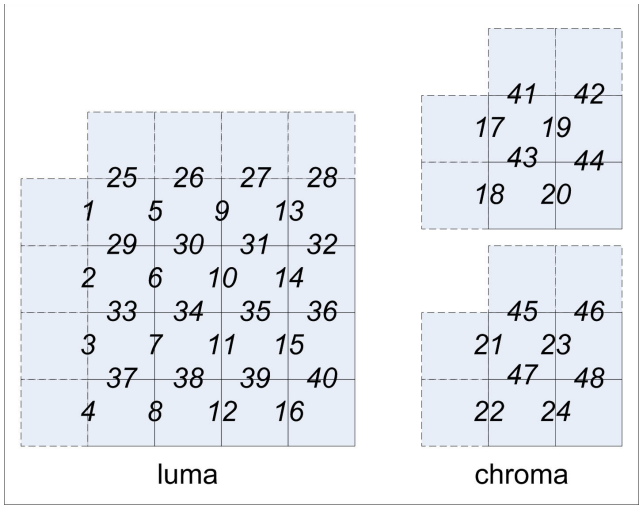

Figure 2.21: Filtering order for a macroblock in H.264/AVC

2.20 and 2.21.

$$Index_A = Min(Max(0, QP + Offset_A), 51) \tag{2.20}$$

$$Index_B = Min(Max(0, QP + Offset_B), 51) \tag{2.21}$$

The actual values of α and β are given from a table which is populated with values derived through empirical tests for producing optimal results for a range of differing content. In general, β(x) is much smaller than α(x). Threshold values increase with QP, and with increasing thresholds, stronger filtering takes place. Conversely, a lower QP value results in less filtering, and a sharper, more detailed final image.

On the slice level, the degree of filtering that will take place can be adjusted through encoder-selectable offsets Offset$_A$ and Offset$_B$. These offset values are used to adjust the QP dependent parameters, α and β. These offset values are transmitted in the slice header syntax, and allow the encoder to override the default quantization parameters, possibly resulting in more optimal decoding.

Filtering Process

In all cases where filtering is carried out, the amount of filtering is adjusted depending on the evaluation of the following two conditions, 2.22 and 2.23.

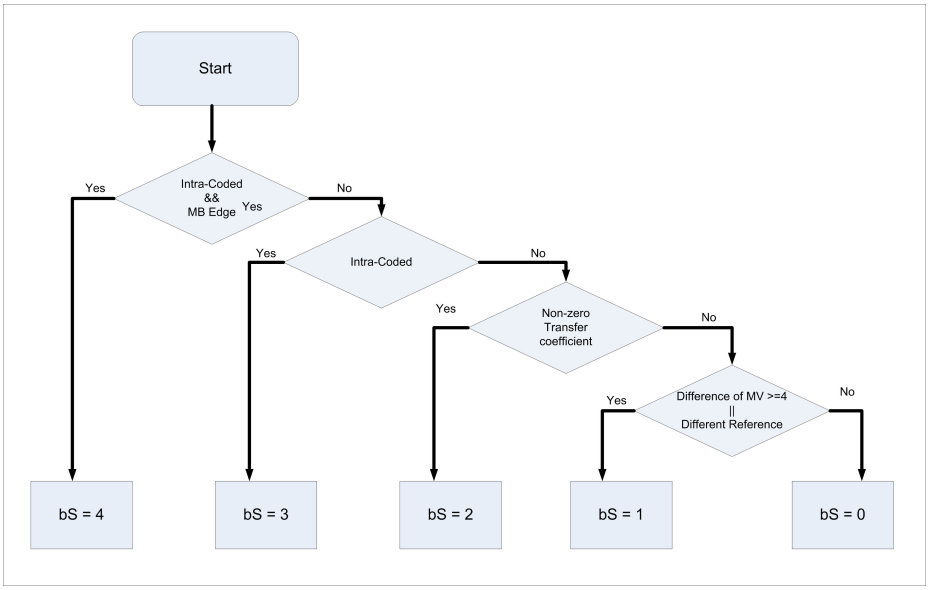

Figure 2.22: Calculation of boundary strength parameters in H.264/AVC

$$|p_2 - p_0| < \beta(Index_B) \tag{2.22}$$

$$|q_2 - q_0| < \beta(Index_B) \tag{2.23}$$

These conditions evaluate the amount of change that exists over interior samples. If there is a great amount of change over these values, then heavy filtering would result in a loss of existing detail. If little change is exhibited across these samples, however, then stronger filtering would result in a continuous smooth image. The next two sections go into the details of filtering for the case of Bs on 1-3, and Bs of 4.

Filtering for Edges with Bs = 1-3

For luminance, the modified values $p_0^{'}$ and $q_0^{'}$ are calculated using 2.24 and 2.25.

$$p_0^{'} = p_0 + \Delta_0 \tag{2.24}$$

$$q_0^{'} = q_0 - \Delta_0 \tag{2.25}$$

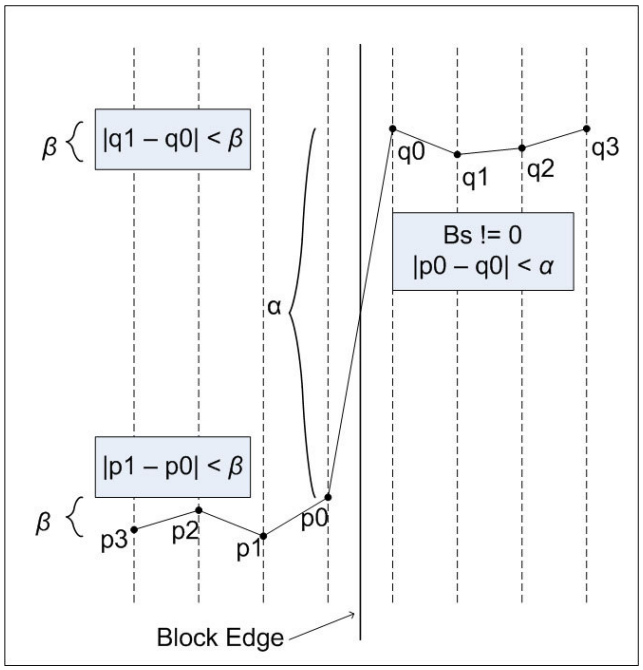

Figure 2.23: Threshold analysis based on α and β

The value of Δ_0 is determined by clipping the value of Δ_{0i}, which is in turn calculated using 2.26

$$\Delta_{0i} = (4(q_0 - p_0) + (p_1 - q_1) + 4) \gg 3 \tag{2.26}$$

The clipping calculations are discussed later. Values for P_1 and Q_1 are modified if the conditions 2.22 and 2.23 hold true, respectively. $P_1^{'}$ is calculated as 2.27 if 2.22 holds true, while $q_1^{'}$ is calculated as 2.28, if 2.23 holds true. The values of Δ_{p1} and Δ_{q1} are calculated in a two step process, similar to Δ_0: these values are the clipped versions of 2.29 and 2.30.

$$p_1^{'} = p_1 - \Delta_{p1} \tag{2.27}$$

$$q_1^{'} = q_1 - \Delta_{q1} \tag{2.28}$$

$$\Delta_{p1i} = (p_2 + ((p_0 + q_0 + 1) \gg 1) - 2p_1) \gg 1 \tag{2.29}$$

$$\Delta_{q1i} = (q_2 + ((q_0 + p_0 + 1) \gg 1) - 2q_1) \gg 1 \tag{2.30}$$

The clipping of 2.26, 2.29 and 2.30 is necessary since too much low-pass filtering would occur otherwise. Clipping, a major adaptivity feature in the filter, is calculated differently for interior and edge samples. The Δ values used for interior samples are clipped the range $-c_1$ to c_1, where c_1 is a value from a 2-dimensional table that is indexed by $Index_A$ and Bs. Final clipping for P_1 and Q_1 is given by 2.31 and 2.32.

$$\Delta_{p1} = Min(Max(-c_1, \Delta_{p1i}), c_1 \tag{2.31}$$

$$\Delta_{q1} = Min(Max(-c_1, \Delta_{q1i}), c_1) \tag{2.32}$$

For edge clipping, the amount clipped is based on c_1 and the evaluation of equations 2.22 and 2.23. c_0, shown in equation 2.33, is first set to to the look-up value of c_1, and then incremented by one for each of 2.22 and 2.23 whose evaluation holds.

$$\Delta_0 = Min(Max(-c_0, \Delta_{0i}), c_0) \tag{2.33}$$

Thus, if the change in intensity is low on either or both sides, stronger filtering is applied, resulting in a smoother final image. If, on the other hand, sharp changes are occurring on the ends, less filtering is carried out, preserving image sharpness.

For chrominance samples, filtering is carried out similar to luminance, except the clipping value c_0 is set to c_1 plus 1. In this way, there is no need to access sample values for p_2 and q_2, and only p_0 and q_0 are modified.

Filtering for Edges with Bs = 4

For filtering with Bs of 4, two different types of filters may be used, depending on sample content. For luminance, a very strong 4- or 5-tap filter, which modifies the edge values and two interior samples, is used if the following condition is met:

$$|p_0 - q_0| < (a \gg 2) + 2. \tag{2.34}$$

If this condition is not met, a weaker 3-tap filter is used which will only modify the edge samples. When 2.23 and 2.34 are both met, filtered values are calculated as shown in 2.35, 2.36 and 2.37 for P values, and 2.38, 2.39 and 2.40 for Q values.

$$p_0^{'} = (p_2 + 2p_1 + 2p_0 + 2q_0 + q_1 + 4) \gg 3 \tag{2.35}$$

$$p_1^{'} = (p_2 + p_1 + p_0 + q_0 + 2) \gg 2 \tag{2.36}$$

$$p_2^{'} = (2p_3 + 3p_2 + p_1 + p_0 + q_0 + 4) \gg 3 \tag{2.37}$$

$$q_0^{'} = (q_2 + 2q_1 + 2q_0 + 2p_0 + p_1 + 4) \gg 3 \tag{2.38}$$

$$q_1^{'} = (q_2 + q_1 + q_0 + p_0 + 2) \gg 2 \tag{2.39}$$

$$q_2^{'} = (2q_3 + 3q_2 + q_1 + q_0 + p_0 + 4) \gg 3 \tag{2.40}$$

For luminance sets in which either 2.23 or 2.34 fails, or for chrominance sets, P and Q are modified as shown in 2.41 and 2.42.

$$p_0^{'} = (2p_1 + p_0 + q_1 + 2) \gg 2 \tag{2.41}$$

$$q_0^{'} = (2q_1 + q_0 + p_1 + 2) \gg 2 \tag{2.42}$$

Chapter 3

Supporting work

Due to design complexity, there has been ample research into the implementation of deblocking filters for H.264/AVC [5], [7], [6], [8], [12], [13], [14]. The main two sources of complexity that have been studied are the high adaptivity of the filter and efficient memory management to handle the irregular access patterns.

3.1 Memory Complexity Management

One source of filter complexity can be attributed to the fact that each pixel needs to be read multiple times, in different data alignments, for the filtering of a single macroblock. Filtering is carried out on the edges of 4x4 pixel blocks, with 16 of these blocks existing within a macroblock, and 8 more from the macroblocks adjacent to the left, and above. Storing macroblocks into memory in a two-dimensional manner corresponding to how they are viewed is most intuitive. This works well for horizontal filtering of vertical edges.

When vertical filtering of the horizontal edges is carried out the data is misaligned. Assuming that 4 horizontal pixels are stored in a line of memory (*i.e.*p3-p0 in horizontal filtering), then each pixel being accessed for vertical filtering is in a different line in memory. This misaligned data, combined with the number of times data needs to be read from and written to memory, has caused many groups to research methods of removing these issues [5], [12], [6].

One method for handling data misalignment is through the use of efficient transpose

circuitry. One transpose design, proposed in [12] and used in [5], rearranges macroblock data for filtering of horizontal edges.

An alternative method for handling data misalignment is to map the pixels into memory such that there is efficient parallel memory access in two directions - both vertical and horizontal. The research proposed in [6] achieves this through the use of 8 dual-port SRAM modules, and a linear shift/rotate mapping scheme which ensures parallel access of 8 pixels. The disadvantage of this technique is that mapping circuitry has to be created, and used for every single write and read. Another disadvantage is the number of memory banks needed for the implementation. The advantage of this design is that no transpose circuitry or datapath is necessary, and memory conflicts are eliminated.

Another approach for reducing memory limitations is to take advantage of data reuse [5], [12], [6], [13]. Since pixel values need to be read multiple times, and intermediate filtering results are needed for later stages, changing the filtering order can allow the filter to reuse data that is recently used sooner in the future. Holding recent data that will be reused close to the filter in local buffers allows for less need to access larger off-chip memory, allowing for less data latency. Various designs have been proposed which attack this problem using different techniques.

The research given in [5] proposes a design in which filtering alternates between horizontal filtering and vertical filtering on a macroblock's 4x4 pixel rows. The filtering order proposed can be seen on the left in Figure 3.1. As a result of this, a row of 4x4 pixel blocks can be stored in local on-chip memory for reuse on the next edge. After pixels are completely filtered, they can be written to main memory for display, or future reference.

Other 2-Dimensional deblocking filter orders have been proposed in [12], and in [13]. These filters alternate between vertical filtering and horizontal filtering at a finer granularity than [5]. For the majority of a macroblock, alternation between vertical and horizontal edge-filtering occurs after each 4x4 edge is filtered, as shown in the filtering order on the right in Figure 3.1. This results in a reduction in local buffer sizes, since data reuse will occur much sooner.

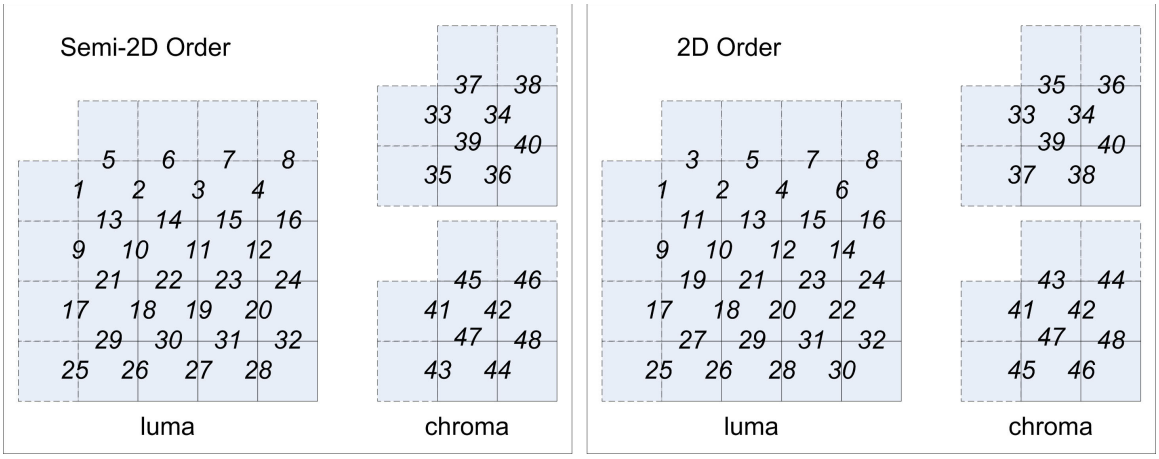

Figure 3.1: Filtering orders for macroblock: left - semi-2D from [5] ; right - 2D from [12]

The design proposed in [6] goes even further, with an architecture which executes vertical and horizontal filtering concurrently. While this greatly reduces the time needed for filtering an entire macroblock, it does require two separate edge-filters: a horizontal and a vertical filter. The data ordering proposed can be seen in Figure 3.2.

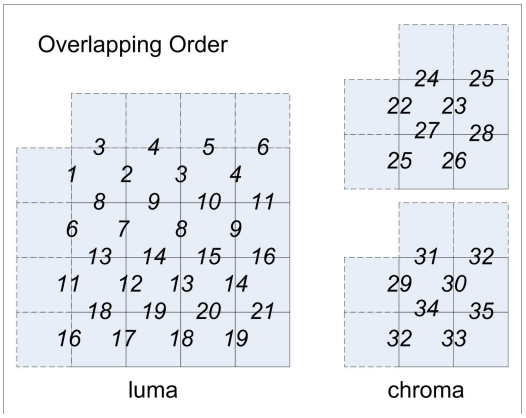

Figure 3.2: Concurrent Filtering order for macroblock in H.264/AVC - [6]

3.2 Filter Complexity

Most research cites the high adaptivity of the filter as the greatest source of complexity [6], [5], [14], [12], [7]. The kernel of the filter's execution is based on conditional execution.

To handle this complexity, varying strategies have been studied, although the majority of which deal with pipelining filter execution.

Pipelining the filtering process allows pixels to be read from data memory, coefficients established, filtering and clipping carried out, and data to be written to memory at a higher available clock speed, since the critical path will be greatly reduced, and execution will overlap, masking latencies associated with the filter architecture. Many designs explore this idea, including [13], [5], [14], and [6]. These designs vary between 4- and 5-stage pipelines. Having the filter pipelined often is coupled to restrictions on possible filtering orders used in order to avoid pipeline data hazards.

One unique technique proposed by [14] suggests implementing multiple pipeline paths each with different filters that are all executed. Each pipe would filter according to a single boundary strength and a select line would be calculated concurrently through the normal evaluation of Bs. This select controls on output mux which selects the output from the proper filter pipeline. While this may speed filtering up, it is not a power-efficient design.

3.3 Motivation for presented research

While many others have come up with novel designs, most do not look in to the process of scaling a design for performance. Previous research presents a design and often compares it against previous research which target different profiles, or use different technologies. When one design targets an ASIC with .35 um technology [7], and another is targeting an FPGA [13], a fair comparison cannot be made.

What is unique about this approach is that multiple designs are studied, all targeting the same technology. A simple and efficient design is first made, and then extra complexity is added through three additional design iterations. The final two design iterations propose a new idea altogether: concurrent filtering of macroblocks. For this reason, the presented work is both unique and important for analysis of deblocking filter's in H.264/AVC.

Chapter 4

Design methodology

Four designs were investigated for this thesis, each having similar basic components. Before exploring the four designs, the sub-units that make up the various deblocking filters are introduced.

4.1 Design Components

Two main hierarchy levels exist within each deblocking filter design. The bottom level components all operate on a single macroblock, while the top level components work on frames. For ease of discussion, the bottom level will be referred to as macroblock-level, while the top level will be frame-level. A description of the sub-units follows.

4.1.1 Macroblock-Level Components

SR Buffer

A buffer is needed in the design to hold results which need to be reused for the filtering of a later edge. This buffer is designed to hold 4 rows of 4-samples. Since the edge filter operates on a single row of samples at a time, a shift register is ideal for holding the data. The interface for this shift register buffer can be seen in figure 4.1. Every design that was investigated makes use of this structure.

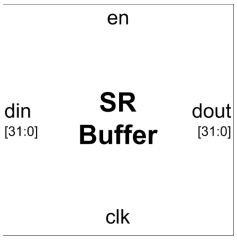

Figure 4.1: Interface for the shift register buffer

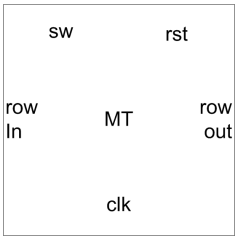

Figure 4.2: Interface for the matrix transpose unit

The shift register buffer takes as input one row of 4 samples, and holds a subblock of sample data (4 rows of 4-sample values).

Matrix Transpose

When filtering vertical edges, one row of memory holds all of the samples on one side of the edge. When filtering horizontal edges, the necessary sample data is located in a single column across 4 rows of memory. A 4x4 matrix transpose unit is used for realigning the data for filtering of horizontal edges. The transpose buffer interface can be seen in figure 4.2, and the RTL design can be seen in figure 4.3.

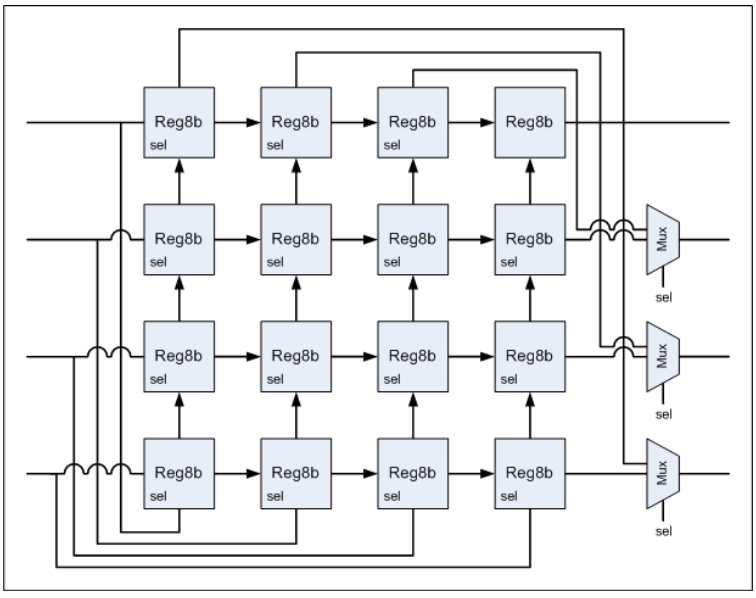

Figure 4.3: RTL for the matrix transpose unit

After shifting in for four cycles (4 4-sample rows of pixels), the transposed pixel block is created by switching the select input.

Edge Filter

The edge filer is identical in all of the investigated designs. This unit, which carries out filtering on a row of 8 pixel values, can be seen in figure 4.4. The amount of filtering that

Figure 4.4: Interface for the edge filter

takes place depends on the input boundary strength *bS*, *indexA*, *alpha*, *beta* and the value of the input pixels. A high level description of this unit can be seen in figure 4.5. Two filtering paths exist in the edge filter as well as a bypass path: one for handling edges with boundary strengths between one and three, and the other for edges with boundary strength of four. This is a purely combinational logic block, providing outputs without a cycle delay. The output of this logic block are the 4 pixels on either side of the filter edge.

Datapath Controller

A deblocking filter controller is necessary for adjusting the datapath to allow for filtering of a macroblock. This unit handles the operation of macroblock memory inputs and outputs, the shifting buffers, the matrix transpose buffers, and the filtering inputs and outputs. This unit consists of a finite state machine which sets all of the control signals that are sent to the data path. The main objective of this block is to feed a full macroblock through the edge filter. The design of this unit varies between the different designs explored.

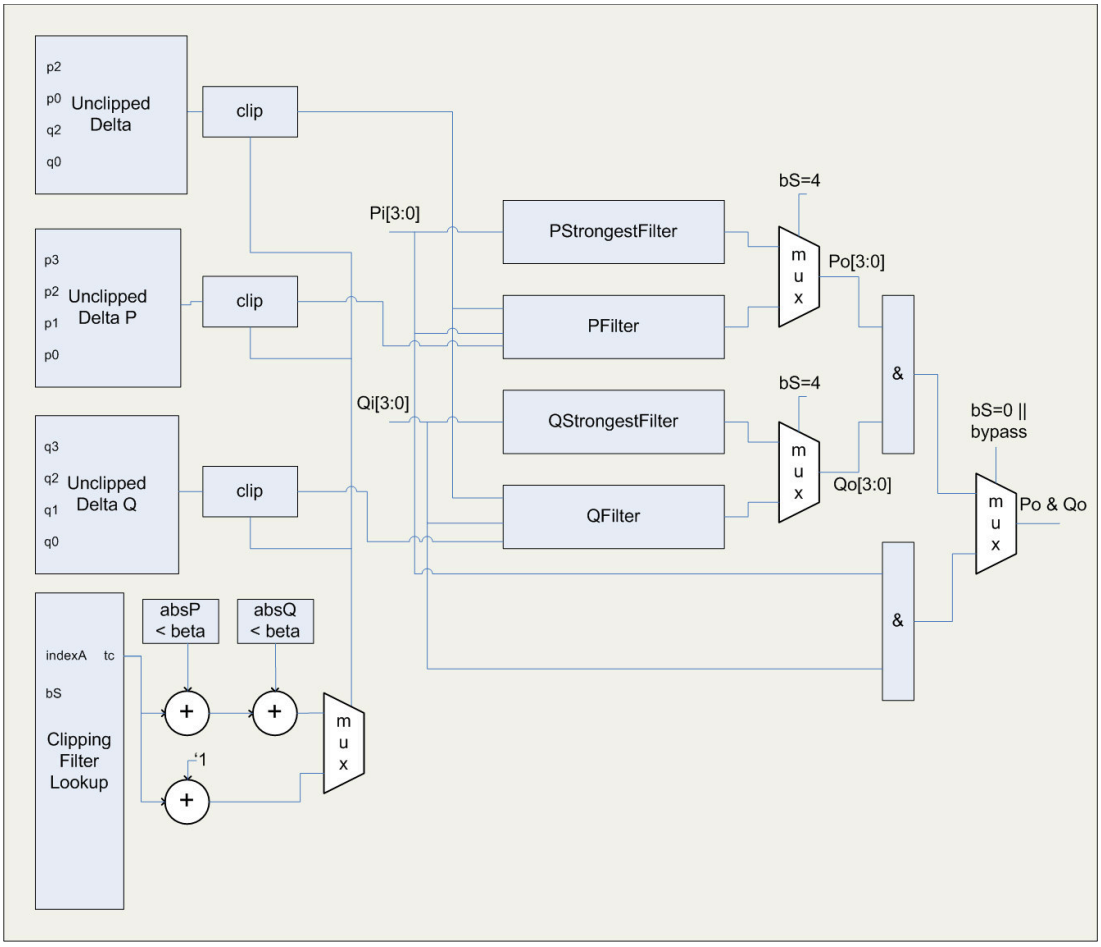

Figure 4.5: RTL for the edge filter

Macroblock Memory

Macroblock memory is used to keep macroblocks closer to the edge filter than would be possible with a full frame. This memory is organized to hold all of the data shown in figure 4.7.

To store all of the macroblock data along with the needed neighboring blocks to the left and above, 160 lines of 32-bit wide memory is needed. The memory layout for the macroblock data is shown in figure 4.8.

For efficient operation, the memory is organized as a dual port read write RAM block.

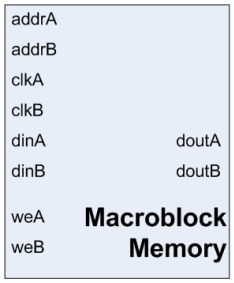

Figure 4.6: Interface for macroblock memory

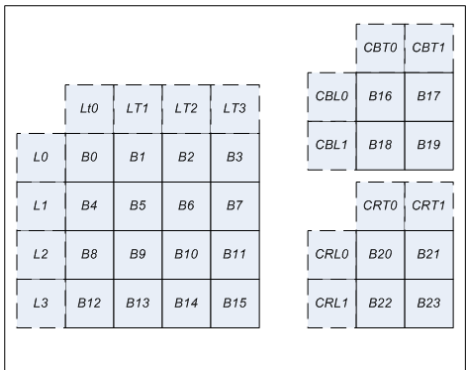

Figure 4.7: Layout for macroblock memory

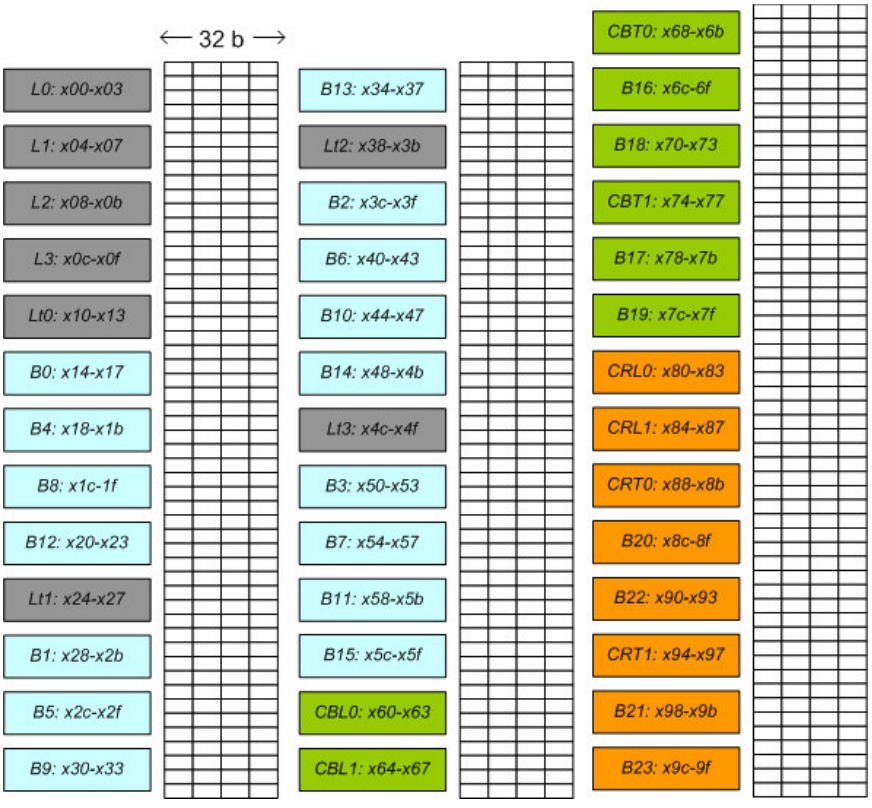

Figure 4.8: Memory map for macroblock memory

4.1.2 Frame-Level Components

Deblocking Filter

The deblocking filter unit is comprised of a combination of all of the macroblock-level components wired together. The exact organization and interface for this unit varies between designs.

Threshold Derivation

The threshold derivation unit is responsible for determining the *indexA*, *beta* and *alpha* parameters for input to the deblocking filter. The interface, seen in figure 4.9, and the implementation are identical for each design.

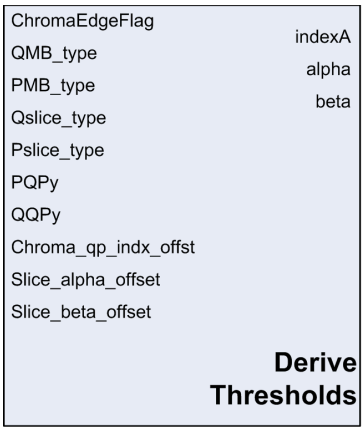

Figure 4.9: Interface for threshold derivation unit

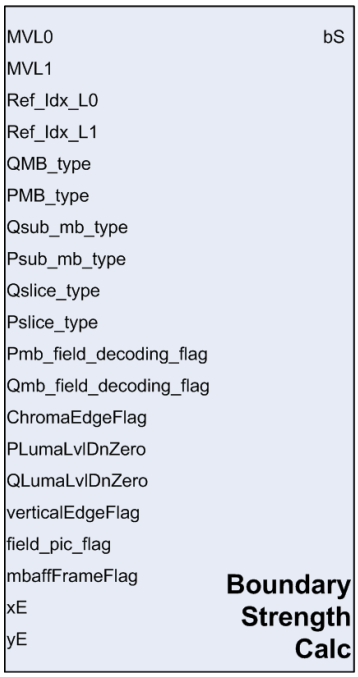

Figure 4.10: Interface for the boundary strength calculation unit

Boundary Strength Calculation

The boundary strength unit is responsible for calculating the boundary strength for each edge within a macroblock. The resulting boundary strength is sent to the deblocking filter. As seen in the interface diagram in Figure 4.10, the boundary strength is based upon many encoding parameters. This unit is identical across all designs.

Macroblock Buffer

It is required that samples from the macroblock to the left and above are available when filtering the current macroblock. As a result, the right-most column and the bottom row of macroblock subblocks are needed for filtering of future macroblocks. The right-most column is needed for the next macroblock, while the bottom row of subblocks are not needed until frame-width macroblocks later, as shown in figure 4.11.

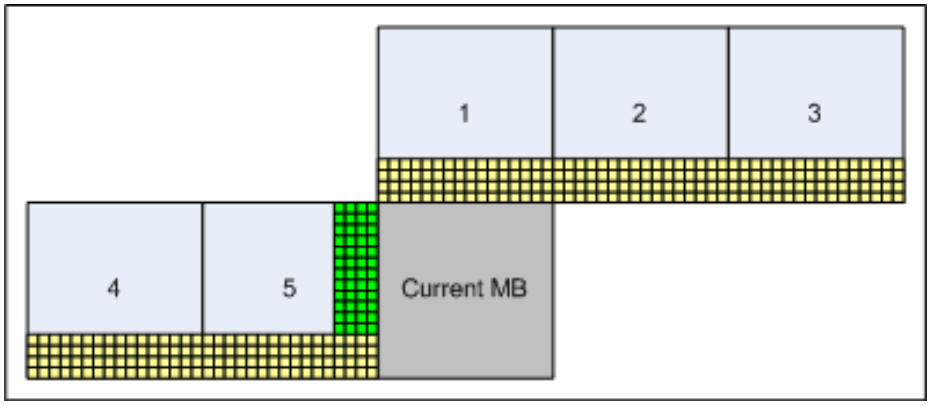

Figure 4.11: Needed macroblock samples

The row buffer is used to hold these subblocks until they are later needed. The size of this buffer is frame-width-in-macroblocks * 16 4-sample rows for luma samples. The interface for the macroblock buffer unit can be seen in figure 4.12. There are three separate internal buffers for the luma, B-Chroma and R-Chroma portions.

The row buffer is implemented as a block of RAM, with an in-pointer and out-pointer, treating it is a queue. The general design of this can be seen in Figure 4.13. This design is repeated three times within the macroblock buffer for Luma, B-Chroma and R-Chroma.

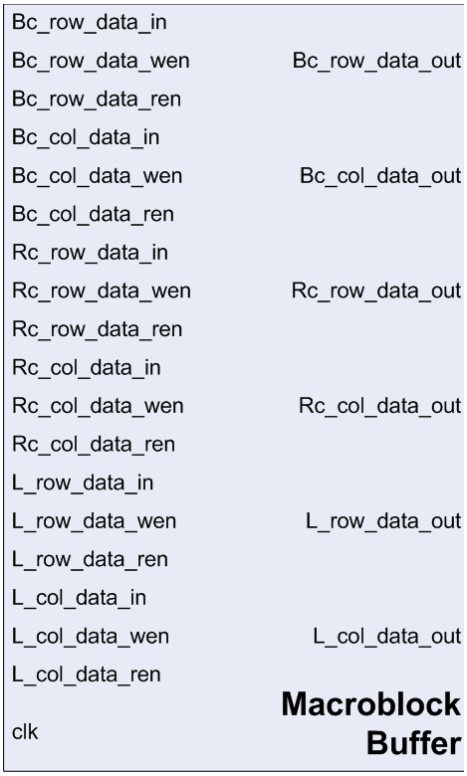

Figure 4.12: Macroblock buffer interface

Wrapper Unit

A separate controller for keeping the deblocking filter filled with new macroblocks is needed, and is implemented within the deblocking filter wrapper unit. Once a new frame is available to be filtered in the decoder, this state machine feeds the frame in to the filter from an outside memory (and from the macroblock buffer), one macroblock at a time, and outputs the filtered frame back to an external memory. Along with the pixel values, header information is also obtained and fed to the boundary strength and threshold derivation units. The wrapper unit also handles data being put into and fed out of the macroblock buffer. Although the interface, shown in figure 4.14, is identical for all designs, the wrapper implementation varies.

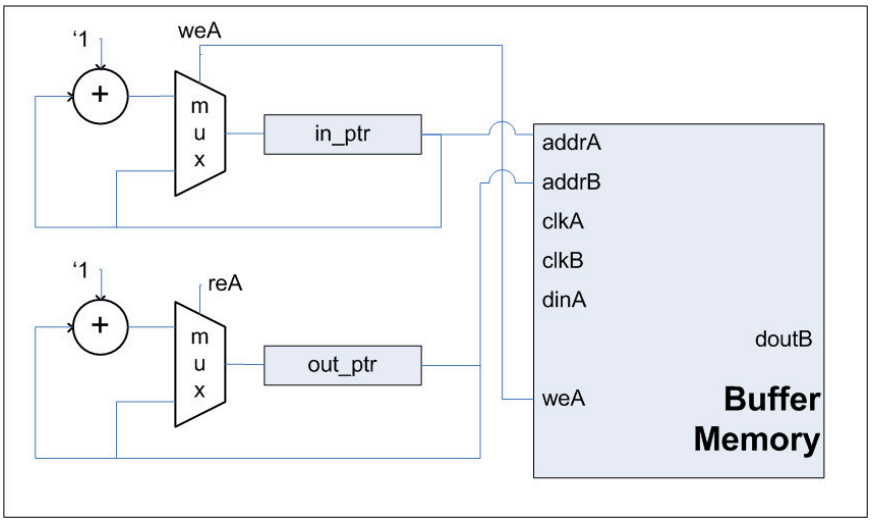

Figure 4.13: Macroblock buffer RTL

Figure 4.14: Interface for the deblocking filter wrapper

4.2 Implemented Designs

4.2.1 Single Serial Filter Design

The first design operates with a single edge filter, filtering a single macroblock at a time. The data ordering used is advanced relative to the order defined in the standard, allowing for a greater temporal locality in the data. Instead of filtering all vertical edges, and then all horizontal edges, the filter alternates between vertical and horizontal edges, as shown in figure 4.15.

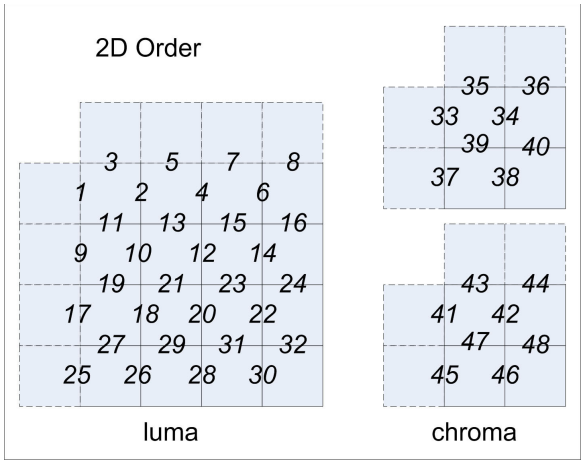

Figure 4.15: Serial edge filter design's data ordering [12]

This order, presented in [12], allows for samples to be reused sooner, reducing the amount of buffering needed in the design. This order also allows for evenly distributed matrix transpose usage, since the filter alternates between vertical and horizontal edges. This design results in an optimal serial filtering time of 192 cycles per macroblock. The filter can be seen in figure 4.16.

The deblocking filter wrapper controller has three main tasks: loading, triggering and storing the filter outputs. This process can be seen in figure 4.17. After storing the resulting macroblock, the controller points to a new macroblock and reiterates until the last macroblock has been filtered. To load the filter, image samples and header information

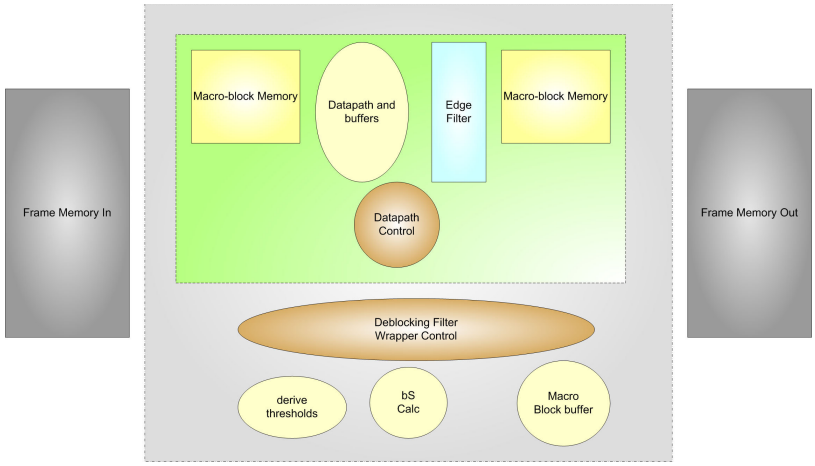

Figure 4.16: Single-serial filter design

is read for the current macroblock from an external memory. Since there is a latency associated with accessing this memory, a preparation state is needed to delay until the data being read is available. In order to obtain sample information from macroblocks to the left and above, the wrapper controller reads these values from the macroblock row buffer. The state machine for the load process can be seen in figure 4.18. The filtering datapath requires four shift register buffers and five matrix transposers. After the data is loaded in the filter, and filtering finishes, the sample data needs to be stored appropriately. Unless the current macroblock is on the right-most column of the frame, the right-most column of samples within the macroblock are stored in the macroblock buffer. These samples will be used during filtering of the next macroblock. Except when the current macroblock is on the bottom row of the frame, the bottom row of samples within the macroblock or stored in the macroblock buffer. These samples will be used as top samples in the following row. There is a separate buffer partition for column and row samples, and within this division there are separate buffers for luma samples, chroma B samples and chroma R samples. If the filtered samples will not be needed again, then they are written out to external frame memory. The frame data store process state machine can be seen in figure 4.19. In parallel with the frame data store, the macroblock header information is buffered in a manner similar to the bottom row and right column of pixels. These parameters are needed when filtering the left and

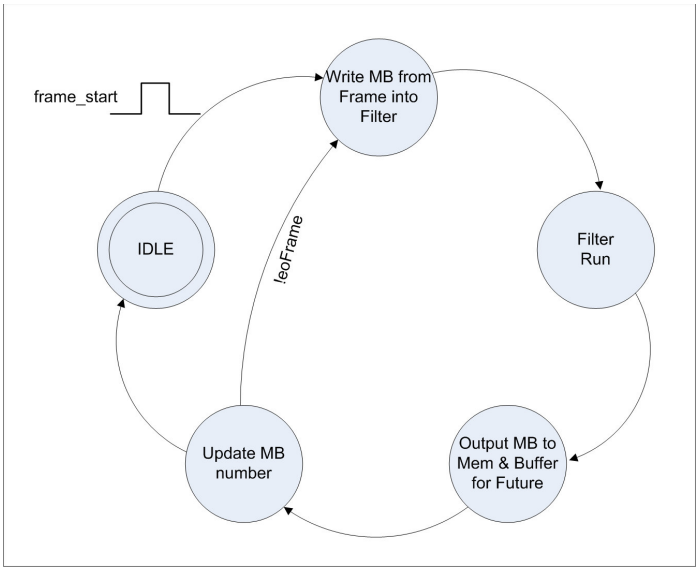

Figure 4.17: Top control unit for the serial edge filter wrapper

top edges of subsequent macroblocks.

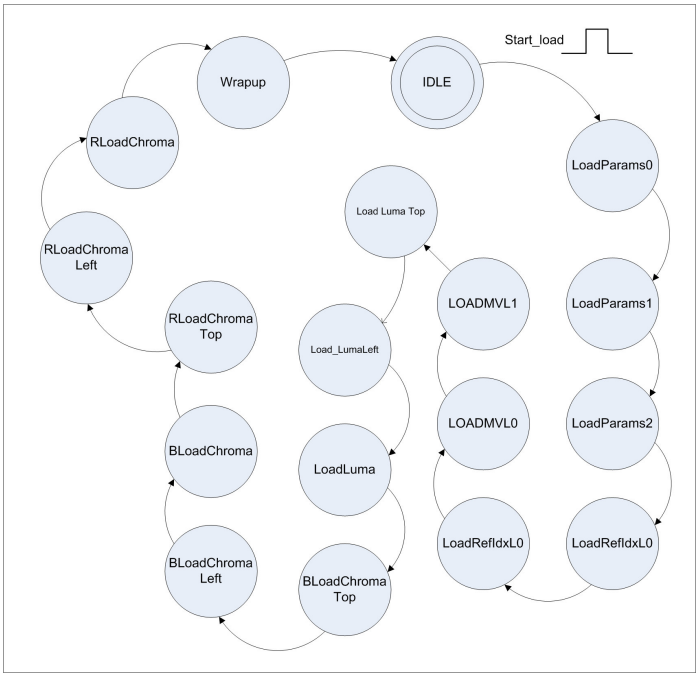

Figure 4.18: Load macroblock control unit for the serial edge filter wrapper

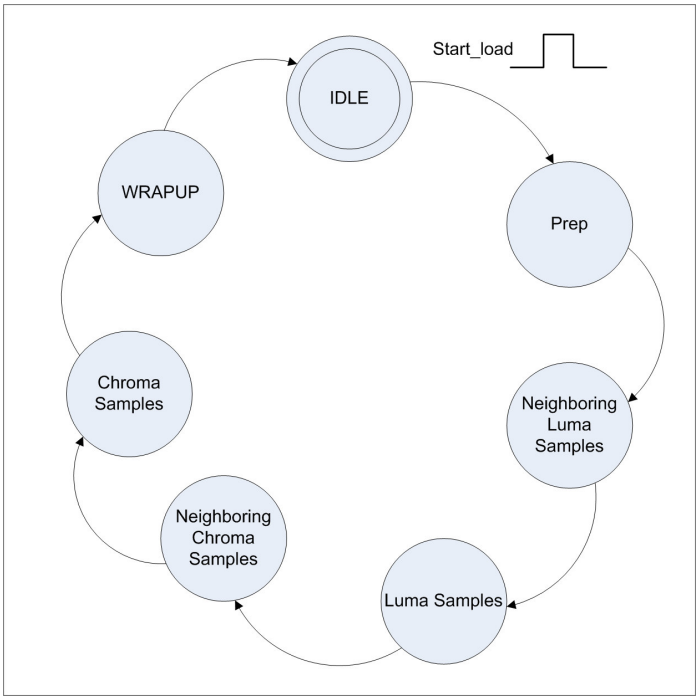

Figure 4.19: Store macroblock control unit for the serial edge filter wrapper

4.2.2 Single Concurrent Filter Design

The second design expands on the first by exploring overlapped block filtering within a macroblock. This design will result in a reduction in filtering time while keeping the amount of time needed for loading and storing a macroblock constant. The data ordering used, shown in 3.2, is the same that was used in [6]. With this data ordering, filtering of a single macroblock can be carried out in 140 cycles, as opposed to 192 cycles in the single edge filter design. The block diagram for this design can be seen in figure 4.20.

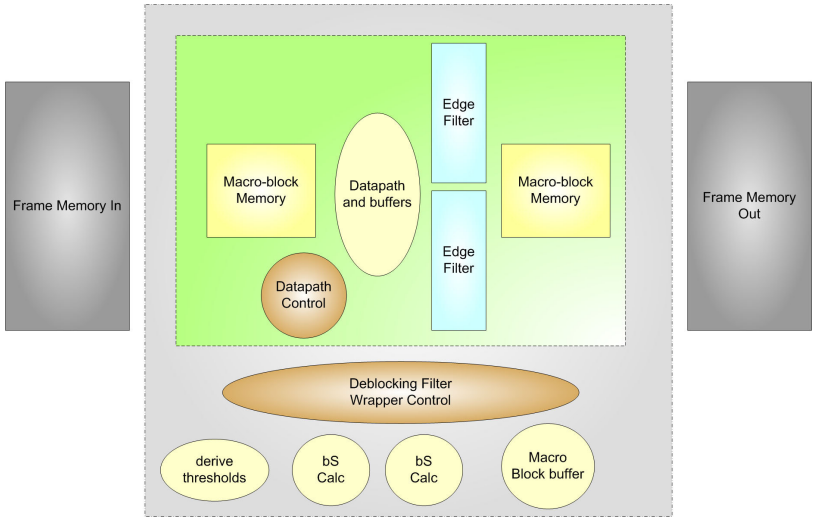

Figure 4.20: Single-concurrent filter design

In order to filter two edges at the same time, an additional edge filter is necessary. With the additional filter, an extra boundary strength calculation unit is necessary. The deblocking filter controller will have significant changes since a more complex datapath that can feed two edge filters is needed. An additional two matrix transpose units are needed for facilitating the two edge filters, for a total of seven. The deblocking filter wrapper needs only slight modifications for wiring the additional boundary strength calculation unit, and the modified interface for the deblocking filter. No changes to the macroblock buffer were needed.

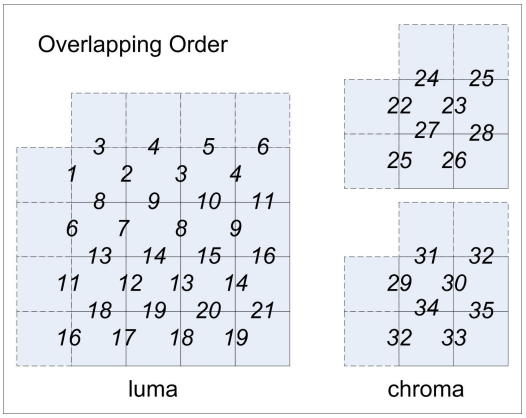

Figure 4.21: Single-conrruent filter design's data ordering [6]

4.2.3 Double Deblocking Filter with Single Edge Filter Design

The third design introduces parallel filtering at the macroblock level. Noting the dependency on the macroblock to the left and above in a frame, a great deal of concurrency can be explored on the frame level. The solution proposed in this thesis can be seen in figure 4.22. The time it takes to filter is approximately cut in half.

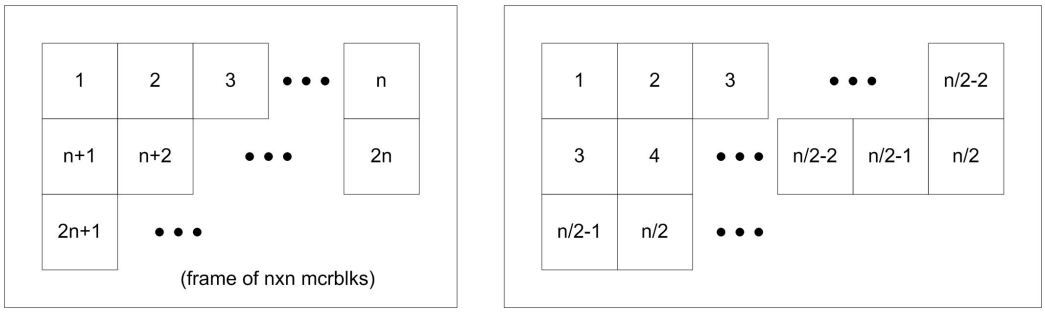

Figure 4.22: Serial and concurrent frame level filtering of macroblocks

The block diagram for this design can be seen in figure 4.23.

The same design from the macroblock-level down is used, with modifications made at the frame-level. Two separate instances of the deblocking filter unit is used from the single edge filter design. A separate boundary strength calculation unit is needed for each deblocking filter. Two modified macroblock buffers are needed to support the two filters. The amount of storage for these buffers is less than that of two of the previous macroblock buffers; the buffer size does not need to be larger since an additional frame-wide row samples is never needed. When the upper filter (that is, the filter which operates to the right of the first macroblock) completes, the bottom row of that macroblock becomes the top of the lower filter (that is, the filter which operates below the first macroblock). This allows for the macroblock buffer to not increase a great deal in complexity.

The top level state machine controlling the concurrent deblocking filters can be seen in figure 4.24.

The read, filter and write portions of the state machine remain constant, only with an additional instance of each in the design. Modifications were needed for the top level state

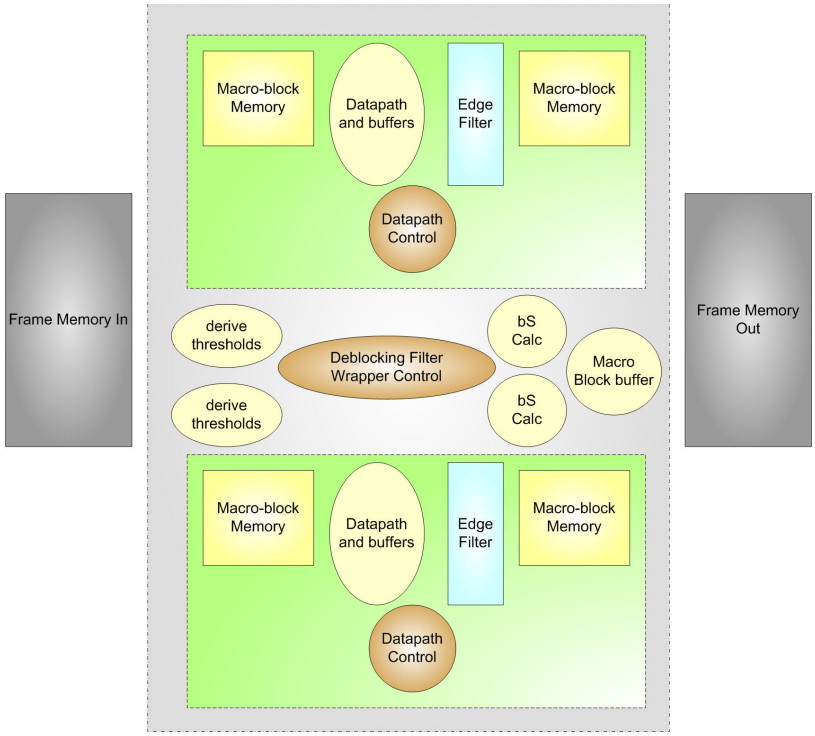

Figure 4.23: Double-serial filter design

machine, seen in Figure 4.24, to handle the single and double filtering. It should be noted that the relative speed up is dependent on whether the number of rows is even or odd. In the case that the there are an odd number of rows, the bottom row needs to be filtered by a single filter serially. When the number of rows is even, however, filtering of the bottom two rows will be overlapped, as are all other row pairs. This design is able to filter a macroblock in effectively 96 cycles.

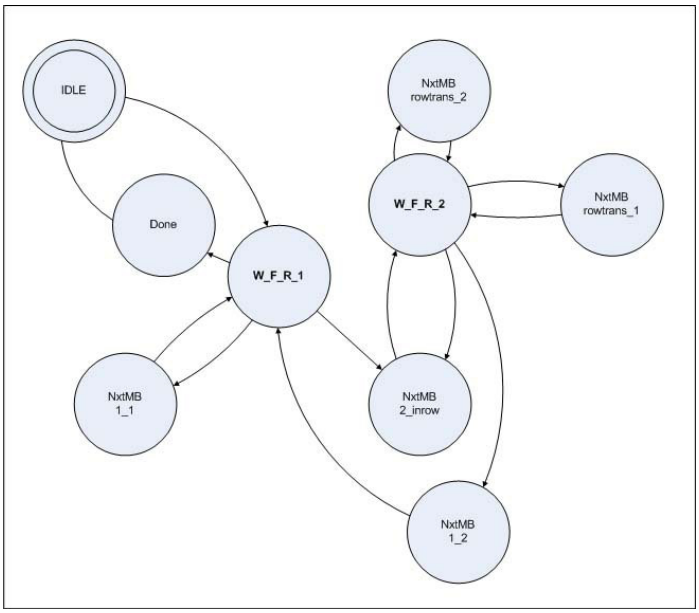

Figure 4.24: Double top finite state machine

4.2.4 Double Concurrent Filter Design

The fourth design explores concurrent filtering at the macroblock level with overlapped filtering of blocks within the deblocking filter. Essentially, two deblocking filters are used which each operate with two edge filters. The block diagram for this design can be seen in figure 4.25.

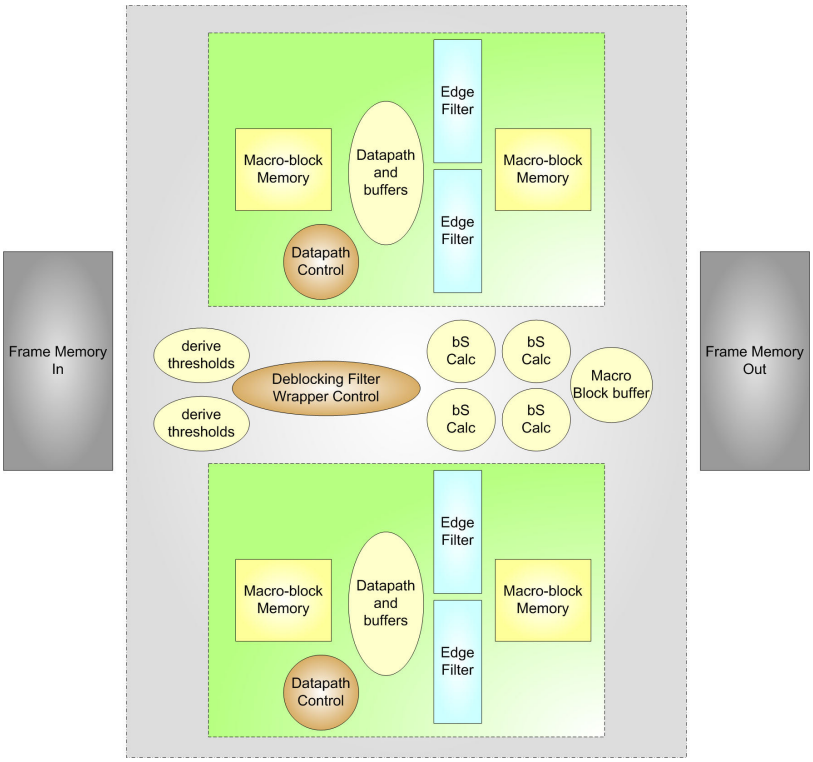

Figure 4.25: Double-concurrent filter design

This design is similar to the double deblocking filter with single edge filters, only with support for the deblocking filter with concurrent edge filters. As a result of this change, two additional boundary strength units are required, for a total of 4. Slight modifications to the top controller are needed to support the extra boundary strength units and the modified interface. Clearly this is the most complex design, and it allows for filtering in the least number of cycles, approximately 70.

Chapter 5

Implementation details

Development of the designs was carried out using primarily ModelSim SE 6.3 and Xilinx ISE 9.1i. Initial design and performance path-finding also utilized Synopsys Design Compiler and Synplicity Synplify Pro 9.0. Area and timing measurements were taken from Xilinx ISE synthesis results. Simulations were run through ModelSim SE on the four separate designs. The accuracy of the filtering process was tested by measuring the residuals for the various designs' output relative to the reference software results. The speed of the filter was measured by analyzing the simulator's waveforms. The time it took to filter an entire frame, filter a macroblock, and read and write a macroblock was noted. From this, the number of cycles necessary for filtering a single macroblock was deduced (both with and without the read and write times).

This chapter first explores the synthesis results for the basic design building blocks, and then gives the resulting specifications and simulation results for each complete design.

5.1 Design component results

5.1.1 SR Buffer

The SR Buffer is implemented as a shift register within the FPGA. SR Buffer's RTL results in the following synthesis specifications.

Synthesis Results	
slice regs	32
slice LUTs	32
gate count	4,352
min clock period	1.216 ns

Four buffers are needed in the single deblocking designs, while eight are needed for the double deblocking designs.

5.1.2 Matrix Transpose

The matrix transpose unit synthesizes with the following results:

Synthesis Results	
slice regs	128
gate count	2,088
min clock period	1.107 ns

In the serial design, five are used, while in the concurrent edge design, seven are used. For the two double deblocking filters, double the number are needed, respectively.

5.1.3 Edge Filter

The synthesis results for the edge filter are given below.

Synthesis Results	
regs	10
slice LUTs	701
gate count	7,639
min delay (unconst)	15.33 ns

The edge filter is the critical core component in every design, located on the critical path. One, two or four filters are used, depending on the design.

5.1.4 Threshold Derivation

The threshold derivation unit has the following synthesis results.

Synthesis Results	
slice regs	32
slice LUTs	124
gate count	984
min delay (unconst)	12.17 ns

Two or four of these units are used, depending on whether it is a single or double deblocking filter design.

5.1.5 Boundary Strength Calculation

Synthesis Results	
slice regs	32
slice LUTs	133
gate count	917
min delay (unconst)	12.74 ns

One, two or four of these units are used, depending on whether it is a serial edge or concurrent edge, or double deblocking filter design, or both concurrent edge and double deblocking filter.

5.2 Design results

The complexity of the controllers in each design and the number of base design components used results in great variation between the synthesis results for each design.

5.2.1 Single deblocking filter with serial edge filter

The smallest and least complex design yields the following synthesis results.

Synthesis Results	
minimum period	16.68 ns
(max. freq.)	59.96 MHz
gate count	1,127,195
slice regs	1,324
slice LUT	6,511
route thrus	458
BRAMS	8
memory	288kB

The resulting filtered image can be seen in figure 5.1. The calculated residual output can be seen in figure 5.2. This image is the difference between the reference software output and the single edge filter output. Since all the implemented filters perform perceptually near-identical, the image results could also be used as reference for the other three designs.

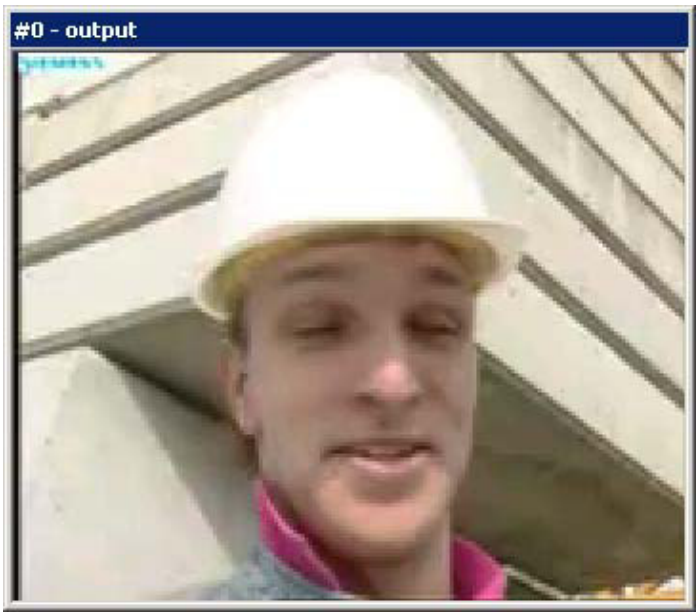

Figure 5.1: Filter output

The residual statistics can be seen in the following table.

Figure 5.2: Filter residual output

Residuals Summary	
Number of coefficients	38,016
Total difference	8,144
Average difference	0.2142
Maximum difference	17
Total residual squared	14,560
MSE	1.0075e-5
RMSE	0.003174
PSNR	98.0985

The residual results and perceptual quality of the image meet the expectations of the deblocking filter.

The following table shows the performance details measured from the simulation waveform. The number of cycles required for writing varies depending on the location within the frame, since sample data from the left and top are not written in the cases where the

macroblock is on the left or top edge, respectively. The recorded value is for the most common case, which is also the greatest number of cycles, where the macroblock is not on the left column or top row.

Simulation Results	
Total cycles	57,078
Reading cycles/MB	181
Writing cylces/MB	200
Filtering cycles/MB	201

5.2.2 Single deblocking filter with concurrent edge filter

The single deblocking filter with concurrent edge filter's synthesis results are seen in the following table.

Synthesis Results	
minimum period	23.21 ns
(max. freq.)	43.08 MHz
gate count	1,138,315
slice regs	1,511
slice LUT	7,605
route thrus	516
BRAMS	8
memory	288kB

The residual statistics for the concurrent edge filter can be seen in the following table.

Residuals Summary	
Number of coefficients	38,016
Total difference	8,144
Average difference	0.2142
Maximum difference	17
Total residual squared	14,560
MSE	1.0075e-5
RMSE	0.003174
PSNR	98.0985

The residual results, and perceptual quality of the image meets the expectations of the deblocking filter.

The following table shows the performance details taken from the simulation waveform.

Simulation Results	
Total cycles	51,930
Reading cycles/MB	181
Writing cylces/MB	200
Filtering cycles/MB	149
Cycles/MB	525

5.2.3 Double deblocking filter with single edge filter

The double deblocking filter with serial edge filter's synthesis results are seen in the following table.

Synthesis Results	
minimum period	17.51 ns
(max. freq.)	57.10 MHz
gate count	2,264,119
slice regs	2,864
slice LUT	14,050
route thrus	1,087
BRAMS	16
memory	576kB

The residual statistics for the double deblocking filter with serial edge filter can be seen in the following table.

Residuals Summary	
Number of coefficients	38,016
Total difference	8,144
Average difference	0.2142
Maximum difference	17
Total residual squared	14,560
MSE	1.0075e-5
RMSE	0.003174
PSNR	98.0985

The residual results, and perceptual quality of the image meets the expectations of the deblocking filter.

The following table shows the performance details taken from the simulation. While individual filtering takes 201 cycles, effectively this number is 105, since two deblocking filters are used in parallel. The effects of this speedup can be seen in the total cycles value.

Simulation Results	
Total cycles	48,102
Reading cycles/MB	181
Writing cylces/MB	200
Filtering cycles/MB	201
Cycles/MB	485

5.2.4 Double deblocking filter with concurrent edge filter

The double deblocking filter with concurrent edge filter's synthesis results are seen in the following table.

Synthesis Results	
minimum period	23.04 ns
(max. freq.)	43.40 MHz
gate count	2,290,343
slice regs	3,356
slice LUT	16,594
route thrus	1,127
BRAMS	32
memory	576kB

The residual statistics can be seen in the following table.

Residuals Summary	
Number of coefficients	38,016
Total difference	8,144
Average difference	0.2142
Maximum difference	17
Total residual squared	14,560
MSE	1.0075e-5
RMSE	0.003174
PSNR	98.0985

The residual results, and perceptual quality of the image meets the expectations of the deblocking filter.

The following table shows the performance details taken from the simulation waveform. Again, it should be noted that the effective filtering time is actually half of what is on the table due to overlapped macroblock filtering.

Simulation Results	
Total cycles	45,242
Reading cycles/MB	181
Writing cylces/MB	200
Filtering cycles/MB	149
Cycles/MB	457

Chapter 6

Testing

Testing was conducted at many points in the design using different methodologies. Early in the design process, testing of individual functional units was carried out. Datapath memory elements were easily verified through testbench simulations. The edge filter was tested initially by applying test patterns to the device under test, and a previously verified implementation presented in [16].

After testing the bottom units, a macroblock-level filter was developed and tested. The deblocking filter controller was first tested for general state transitions. Next, the controller-datapath system was tested by feeding a macroblock through the filter from an input macroblock memory, to an output macroblock memory. For enhanced visibility in testing, the memory contents were initialized to match their current address (i.e. - address 0 held 0, while address 453 held 453). While this handled much of the datapath coverage, it did not allow for proper filter function to be exercised. Both pre- and post-synthesis simulations were carried out in this manner

The interface for the macroblock level design was made identical to [16] to leverage preexisting testing structures, seen in figure 6.1. This allowed for a more thorough testing of the deblocking filter, since actual frame data was used.

Interfacing this test setup with a post-synthesis macroblock-level deblocking filter was not directly possible, and was not carried out. The same methodology for testing this hierarchy of the design was used for the concurrent edge filter.

Next, the frame-level hierarchy was tested. Individually, the threshold derivation unit

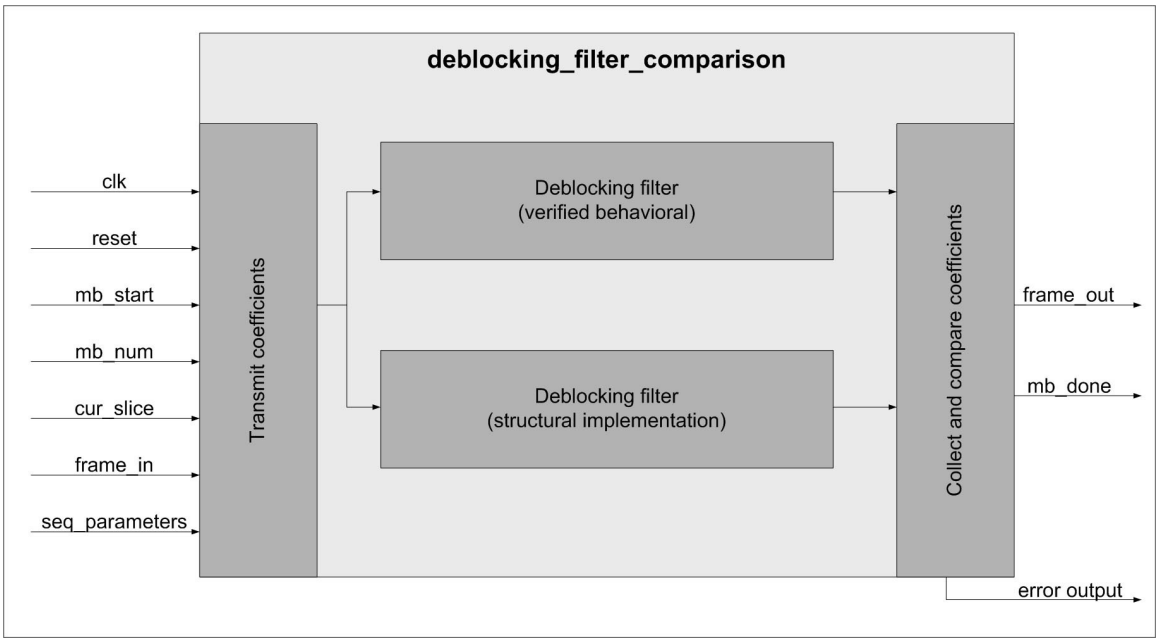

Figure 6.1: Initial setup for testing functionality of deblocking filter - from [16]

and boundary strength calculation units were tested, comparing results to known-working designs from [16]. The top-level controller was also individually tested for proper state transitions.

Due to needed modifications and enhancements in the design, the previous test harness could no longer be used for testing the complete design. The enhanced testing architecture that was used can be seen in figure 6.2

In order to facilitate this, modifications were made to the H.264/AVC standard's reference software [2]. Frame and parameter data were packed and written to a file, which was later used to write in to the design's frame memory. With this setup, the deblocking filter could be compared directly to the purely software implementation for verification. The frame-level testing was carried out using pre-synthesis RTL, and then post-synthesis netlists. This process was used for both the single and double deblocking filter designs.

When errors occurred during the verification process, various forms of debug were used: viewing the resulting image and searching for patterns; viewing the residual image (deduced from the reference software output); looking for patterns in the output memory;

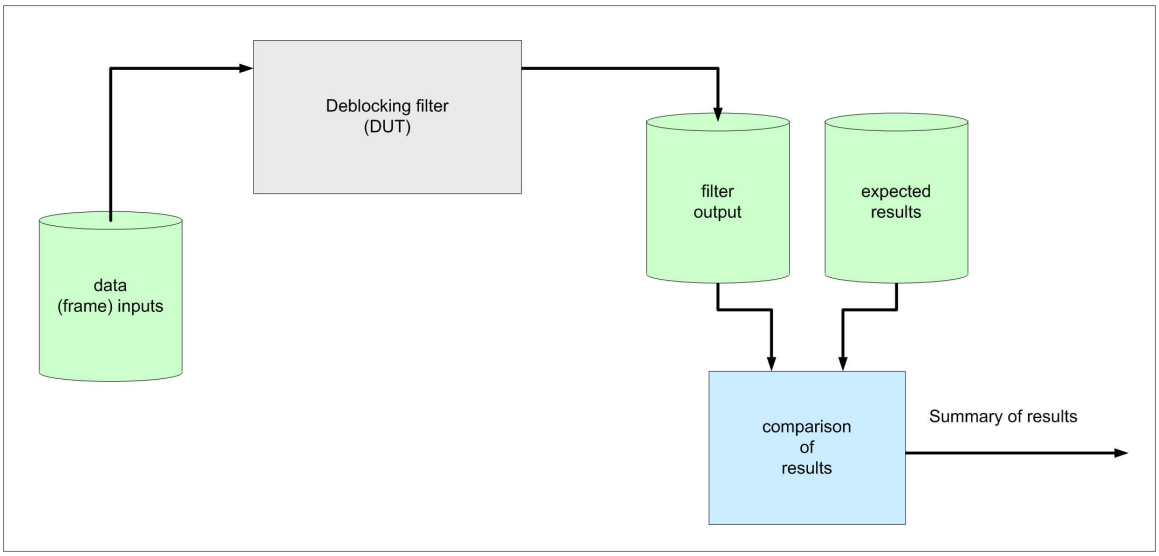

Figure 6.2: Architecture for testing advanced deblocking filter designs

and looking at the simulation waveform. While the first three techniques are very tedious, they are often necessary. The hardest bugs to resolve occurred during post-synthesis simulations, where informative waveforms were not available (because translated and mapped design nets and structures are renamed below the interface).

In general, zero residual results were not expected in the hardware implementation. For a deblocking filter, it is most important to be perceptually near-identical to the original image. For this reason, in testing, residuals are recorded between the design under test and the software output. The variance is measured, and errors are flagged when this variance is too great.

Chapter 7

Analysis of results

7.1 Proposed designs

As expected, the effective gate count for each design steadily increased as the complexity of the design was scaled. The concurrent edge filter design was greater in size than the serial, since an extra edge filter and a more complex datapath was needed. As expected, the double deblocking filters were approximately twice as large as the original designs. A summary of the design sizes (in effective gate count) can be seen in the following table.

Single-serial	Single-concurrent	Double-serial	Double-concurrent
1,127,195	1,138,315	2,264,119	2,290,343

The majority of the design effort in this research was based upon speeding up the filtering process. Through careful testing and design optimization, this was achieved successfully. The resulting deblocking filter performance for each of the scaled designs can be seen in figure 7.1. The baseline for the speedup in filtering time is the time for filtering in the single deblocking filter with serial edge design. The most complex design, the double deblocking filter with concurrent edge, has the greatest speedup, 2.66.

While these performance numbers scale well, they only consider the amount of time it takes to filter macroblocks, ignoring the time it takes to read and write a macroblock in to and out of the filter. For the complete deblocking filter, the performance gains are much less, as shown in figure 7.2.

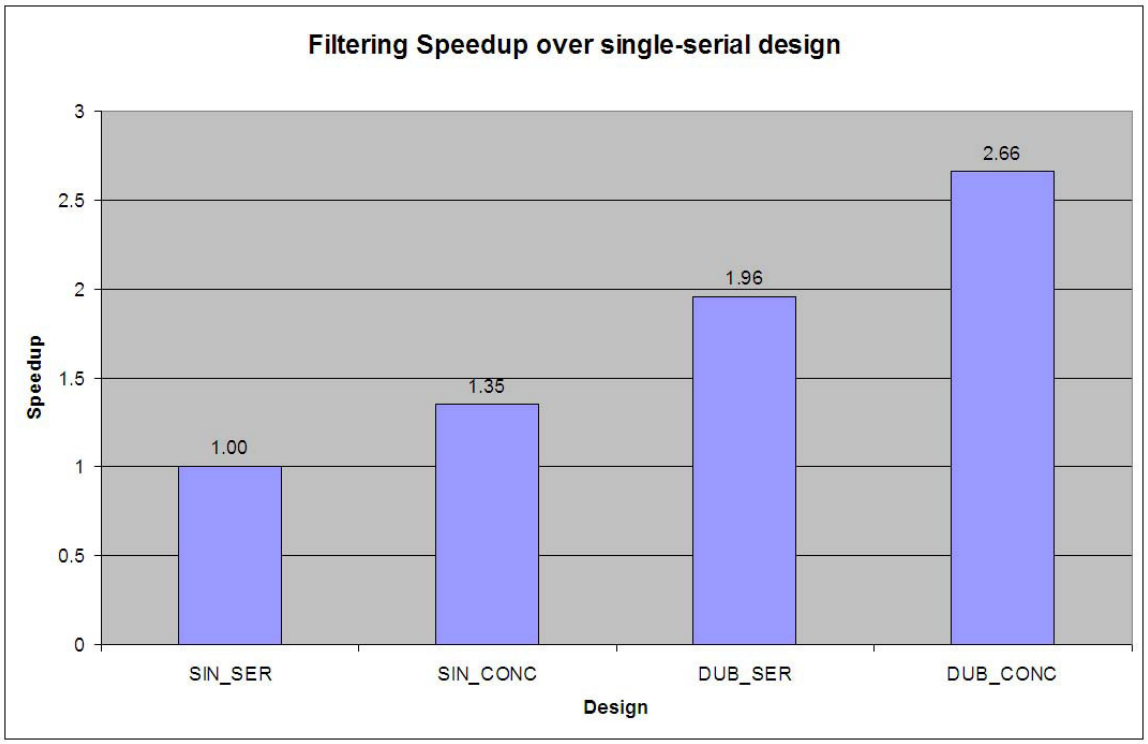

Figure 7.1: Speedup in filtering time relative to single-serial design

When applying computer architect Gene Amdahl's theorem to these results, the total performance gains make sense. Amdahl's law is stated below, where P_K represents the percentage of time that can be improved through parallelization, and S_k represents the amount of speedup allowable for this portion.

$$\frac{1}{\sum_{k=0}^{n} \frac{P_k}{S_k}}$$

In the presented designs, the filtering process only accounts for a third of the necessary execution. Assuming this results in a speedup of 2, then the maximum expected speedup will be 1.198. This coincides with the results shown in figure 7.2.

These results lead to an investigation in to the possible speedup achievable by reducing the time necessary for reading and writing. For the single deblocking filter designs, a minor modification to the top-level controller was made to allow for concurrent reading and writing of macroblocks. Instead of waiting for the previous macroblock to be read out,

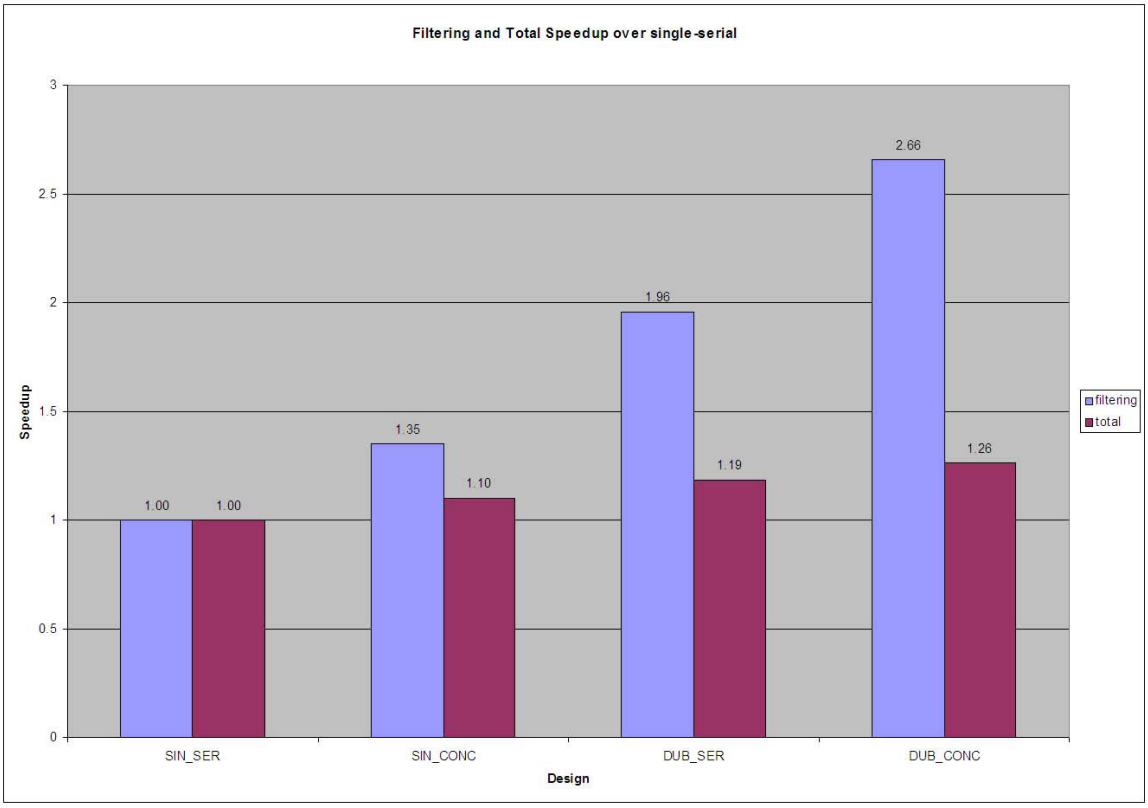

Figure 7.2: Speedup in total filtering time, including reading and writing, relative to single-serial design

the writing of the current macroblock is carried out at the same time. This would effectively cut 2/3 of the execution time in half, allowing for a maximum speedup of 1.49. The results of this design enhancement, along with the original four designs, can be seen in figure 7.3.

Reducing this load and store time has a dramatic impact on the overall filter performance. The cost of this performance increase is the addition of one extra register in the design. This is very impressive, considering the amount of complex design that was previously needed in the filter to achieve even less overall speedup. Synthesis results are nearly identical to the original designs, while filtering quality is identical.

In order to investigate the effectiveness of the design scaling, the performance of each design with respect to its gate count was investigated. Figure 7.4 shows the performance for each design divided by their respective gate counts. The performance is calculated as

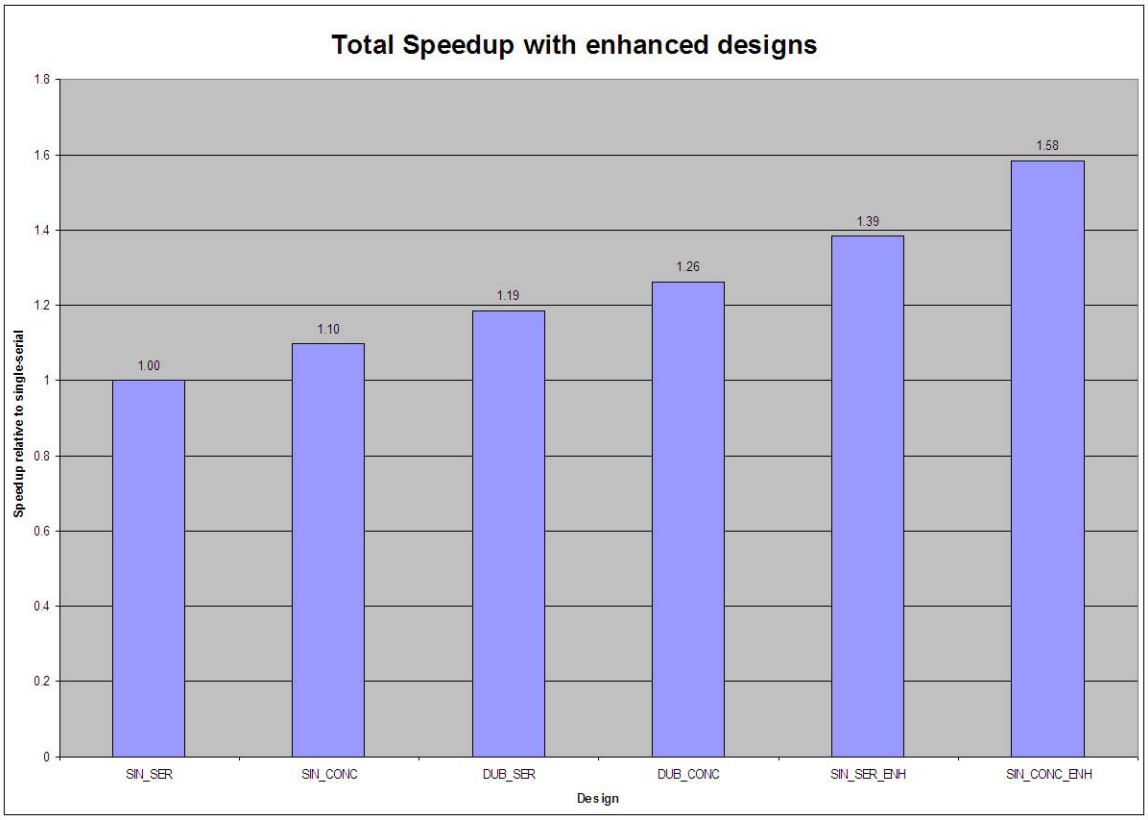

Figure 7.3: Speedup in total filtering time, including reading and writing, relative to single-serial design

cycles per macroblock, and these are normalized to the performance of the single-serial.

Due to the minimal complexity increase needed for enhanced single-deblocking filter designs, these show the best relative speedup. The complexity needed for the double-filter designs are double that of the single-filter designs, and result in a less than 2x speedup. As a result, the performance per gate is the least of the resulting designs. If these designs were enhanced in a similar manner as the single-filter implementations, the results would be improved, but still not at the same level as the single-filter designs.

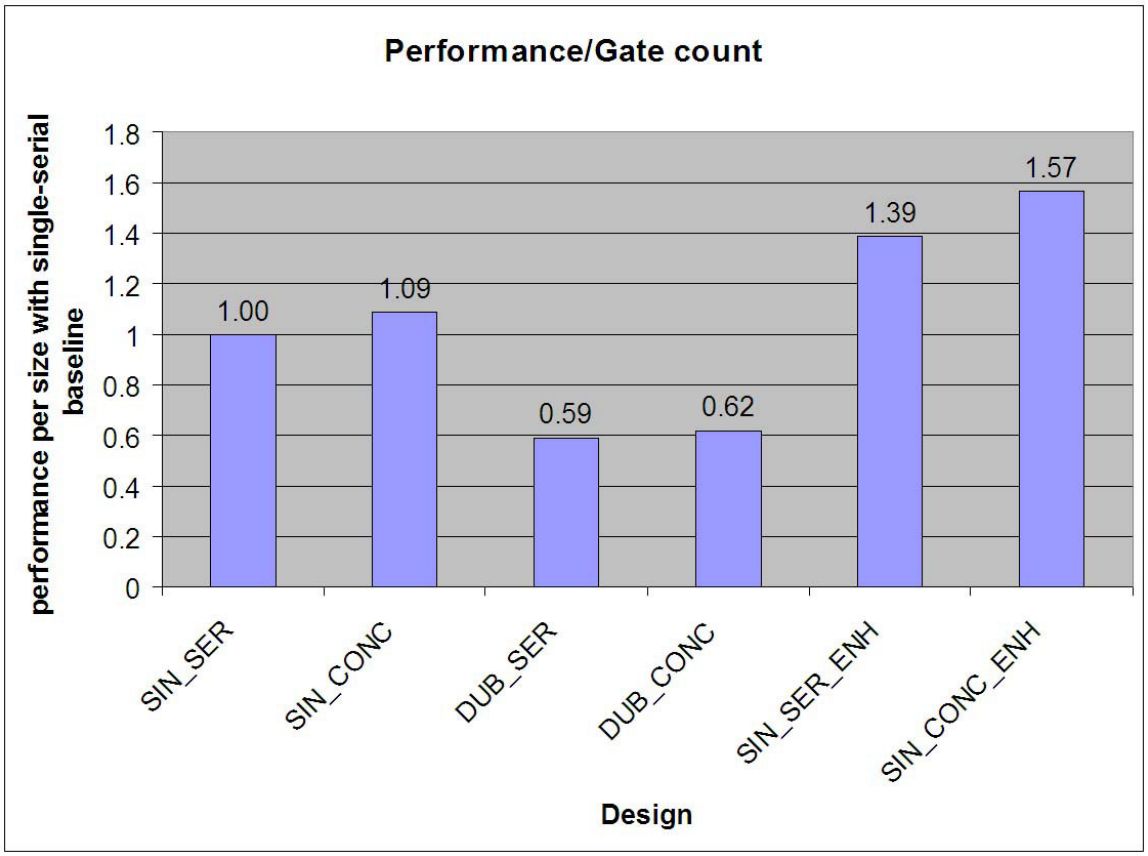

Figure 7.4: Speedup for each design divided by gate count

7.2 Comparison to other research

The minimum clock period generated by synthesis was lower than expected, allowing maximum frequencies on the range of 30 MHz. While the number of clocks needed per macroblock is low, 30 MHz is too slow for applying this filter to larger formats at higher frame rates. This result is due to a long critical path through the macroblock-level of the designs. To mitigate this, an extra pipeline stage or two would be needed in the datapath in the macroblock-level controller. The critical path in every design is at this hierarchical level, from the input to the boundary strength unit to the input of the filtered macroblock-memory. While this would be a small architectural adjustment, the changes necessary in the controller in the RTL would be extensive and prohibitive in this research's scope.

For fair comparison of the designed architectures, and not specific synthesis details,

Design	Cycles/MB (total)	Cycles/MB (filter)
Single-Serial	576	200
Single-Concurrent	524	148
Double-Serial	486	102
Double-Concurrent	457	75
Single-Serial Enhanced	416	200
Single-Concurrent Enhanced	364	148
[13]	214/246	192
[5]	–	192
[7]	566	192
[12]	446	–

Table 7.1: Performance relative to published research

this research will be compared to other designs based on the number of cycles necessary per macroblock. This performance metric is used across the other published research ([13], [5], [7], [12]). A summary of the performance for various designs is shown in the table below.

From this table, it can be seen that the architectures presented in this research are competitive with the other published research.

Chapter 8

Conclusions from work

A great deal was learned in studying the development of a deblocking filter through four (plus two additional enhanced) designs, each scaled with increased complexity. The complexities associated with various deblocking filter techniques were investigated, and a new deblocking filter design approach, the introduction of multiple macroblock-filters, was implemented. Having designed all four components has the added benefit of being able to directly compare architectural ideas. When comparing results against other research, performance measurements and analysis are not always consistent, and do not yield direct comparisons.

While the maximum operation frequency was not comparable to other published research, the original single-filter with serial edge design is comparable to other published research designs. Using this architecture as a baseline, it is possible to analyze how other designs could be relatively improved with the introduction of new techniques.

Future work should investigate increasing the frequency of the given designs. This should be achievable by further pipelining the macroblock-level designs, thus reducing the number of gate delays in the critical path.

Another area of research in the future should be implementing read and write overlap in the double-deblocking filter designs. This improvement alone should allow for a speedup of 1.89 and 2.1 over the single-serial design for the serial edge and concurrent edge filters, respectively.

Further reducing the effects of the read, write and filter time should also be investigated

for every design. By starting filtering as soon as the first samples are ready in the input macroblock, and starting the read process as soon as results are ready, the overall number of cycles needed per macroblock should decrease significantly. While this may require some increased design challenges, the hardware complexity should increase very little.

Bibliography

[1] H.264/MPEG-4 AVC Video Compression Tutorial. Available at http://www.lsilogic.com.

[2] H.264/MPEG-4 JM Reference Software package v12.2. Available at http://iphome.hhi.de/suehring/tml/.

[3] A. Fischer. IMAGO Proposals - European Federation of Cinematographers. Available at http://www.imago.org, 2006.

[4] Y. W. Huang, T. W. Chen, and B. Y. Hsieh. Architecture design for deblocking filter in H.264/JVT/AVC. *Proceedings - International Conference of Multimedia and Expo (ICME03)*, 1:693 – 696, 2003.

[5] G. Khurana, A. A. Kassim, T. P. Chua, and M. B. Mi. A pipelined hardware implementation of In-loop Deblocking Filter in H.264/AVC. *IEEE Transactions on Consumer Electronics*, 52(2):536 – 540, 2006.

[6] L. Li, S. Goto, and T. Ikenaga. A highly parallel architecture for deblocking filter in H.264/AVC. *IEICE Transactions on Information and Systems*, E88(7):1623 – 1628, 2005.

[7] L. Li, S. Goto, and T. Ikenaga. An efficient deblocking filter architecture with 2-dimensional parallel memory for H.264/AVC. In *ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*, pages 623–626, New York, NY, USA, 2005. ACM Press.

[8] P. List, A. Joch, J. Lainema, G. Bjntegaard, and M. Karczewicz. Adaptive Deblocking Filter. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):614 – 619, 2003.

[9] T. M. Liu, W. P. Lee, and C. Y. Lee. An area-efficient and high-throughput deblocking filter for multi-standard video applications. *Proceedings - International Conference on Image Processing, ICIP*, 3:1044 – 1047, 2005.

[10] I. E. G. Richardson. *H.264 and MPEG-4 Video Compression – Video Coding for Next-generation Multimedia*. John Wiley & Sons Ltd, 2003.

[11] I. E. G. Richardson. H.264/MPEG-4 Part 10 White Papers. Available at http://www.vcodex.com/h264.html, 2004.

[12] B. Sheng, W. Gao, and D. Wu. An Implemented Architecture of Deblocking Filter for H.264/AVC. *Proceedings - International Conference on Image Processing, ICIP*, 1:665 – 668, 2004.

[13] S. Shih, C. Chang, and Y. Lin. A near optimal deblocking filter for H.264 advanced video coding. *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, 2006:170 – 175, 2006.

[14] M. Sima, Y. Zhou, and W. Zhang. An efficient architecture for adaptive deblocking filter of H.264/AVC video coding. *IEEE Transactions on Consumer Electronics*, 50(1):292 – 296, 2004.

[15] G. J. Sullivan, P. N. Topiwala, and A. Luthra. The H.264/AVC advanced video coding standard: overview and introduction to the fidelity range extensions. volume 5558, pages 454–474. SPIE, 2004.

[16] T. Warsaw and M. Lukowiak. Architecture design of an H.264/AVC decoder for real-time FPGA implementation. pages 253–256. ASAP, 2006.

[17] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan. Rate-constrained coder control and comparison of video coding standards. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):688 – 703, 2003.

[18] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560 – 576, 2003.